ARPA ORDER NO. 2223
ISI/SR-79-14

1978
Annual Technical Report

July 1977 - September 1978

A Research Program in Computer Technology

Prepared for the

Defense Advanced Research Projects Agency

UNIVERSITY OF SOUTHERN CALIFORNIA
INFORMATION SCIENCES INSTITUTE
4676 Admiralty Way Marina del Rey California 90291

78 04 [illegible] 3

COCKPIT COMPUTATION

Sponsor: ONR **Project leader: Danny Cohen**

The purpose of this work is to help ONR identify problems with and propose research vectors in the application of modern information processing techniques to the information systems in the cockpit of modern aircraft. The work is focused on the information aspects of the modern aircraft man/machine interface. The traditional application of computer graphics to these problems is reviewed and discussed, and new pilot-oriented applications of computer graphics are proposed. In addition, several research directions are proposed, both in new information presentation technology and in the application of the upcoming VLSI technology to these problems.

COMPUTER GENERATED IMAGERY FOR FLIGHT SIMULATION

Sponsor: ONR **Project leader: Danny Cohen**

The purpose of the work is to review the state-of-art of the Computer Image Generation (CIG) technology in order to help the Navy assess its application to pilot training in the mid 1980s. The review includes not only the current state of CIG technology, but also the current trends that can be used for extrapolation of future progress.

MICROPROGAM VERIFICATION

Sponsor: RADC **Project Leader: Stephen D. Crocker**

The usual role of microcode is to implement a complicated instruction set on a simple host machine. If the intended behavior of the instruction set can be formalized, it is reasonable to ask whether the microcode is correct with respect to the formal specification. This project is developing a verification system for proving the correctness of microcode. Inputs to the system include formal descriptions of the target instruction set and the host machine, a listing of the microcode and proof directives that guide the verification system to build up a proof. The language for writing the formal descriptions is ISPS, a variant of Bell and Newell's instruction set processor notation. The Air Force Fault-Tolerant Spaceborne Computer (FTSC), the primary example being used to drive the construction of the verification system, is a microcode machine with about 750 very wide microinstructions. The target machine has 32-bit instructions that perform the usual complement of arithmetic and logical operations, including floating-point arithmetic. Formalization of the floating-point operations is one of the more challenging subtasks of this effort.

This document describes activities conducted at USC/Information Sciences Institute under the sponsorship of the Defense Advanced Research Projects Agency's Information Processing Techniques Office (ARPA-IPTO). The projects at ISI supported by other sponsors are listed below. If you desire information about these areas, please contact the project leader specified.

COMPOSITION OF MULTIPARAGRAPH TEXT

Sponsor: NSF

Project Leader: William C. Mann

This project is developing new methods for autonomous text composition by machine, with the focus at the larger-than-sentence level. An experimental system called KDS (Knowledge Delivery System) has been developed as a testbed for methods of text composition from computer-internal knowledge representations such as semantic nets. Recent progress with KDS includes a new expressive paradigm that does not rely on dividing the semantic net into sentence-sized pieces; a problem-solving text organizer that delegates part of its work to other processes; explicit principles for ordering and grouping concepts for expression; rules for aggregating small expressions into larger, more natural ones; and a novel text improvement method based on a rule-competition-oriented variant of the classic hill-climbing algorithm. The ideas in KDS have been tested on two principal domains: description of the structure and use of a computer message system and description of a multi-actor, multi-contingency policy on how to respond to indications of fire in a computer center.

COMPUTER COMPREHENSION OF NATURAL LANGUAGE

Sponsor: NSF

Project Leader: William C. Mann

Despite its great social significance, human communication remains a complex, poorly understood process. Particularly obscure is comprehension, the process by which a sequence of words affects the cognitive state of the hearer. Human-computer interaction could be made easier and more effective if computers could comprehend strings of symbols in more human-like ways. Our prior research has created a model of human communication as a kind of goal-pursuit activity. The model represents four principal levels of knowledge in communication: goals being pursued, speech acts used to pursue goals, propositions exchanged while performing speech acts, and linguistic symbols used to convey all of these. This project, still in its early stages, will build an experimental system called DCS (Dialogue Comprehension System), which will read and follow transcripts of actual human dialogues. In particular, it will contain processes that can recognize the establishment, pursuit, and satisfaction of individual goals, including representation of state changes in the communicating individuals.

TRANSFORMATIONAL IMPLEMENTATION

Sponsor: NSF

Project leader: Robert Balzer

This project is investigating the facilities needed to support an alternative programming paradigm based upon an interactive system in which the programmer designs the optimization of a high-level formal specification and the system implements that design by applying corresponding transformations from a catalog. In this paradigm, the programmer's responsibility would be to determine how to optimize the formal specification, while the system's responsibility would be to ensure the validity of the resulting implementation.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: ISI/SR-79-14	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): 1978 Annual Technical Report: A Research Program in Computer Technology.	5. TYPE OF REPORT & PERIOD COVERED: Annual Technical Report. 1 Jul 1977 - Sept 1978
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): ISI Research Staff; Keith W. Uncapher	8. CONTRACT OR GRANT NUMBER(s): DAHC 15-72-C-0308; ARPA Order 2223
9. PERFORMING ORGANIZATION NAME AND ADDRESS: USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90291 407952	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order #2223
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: Feb 1979; 13. NUMBER OF PAGES: 162
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): ------ (12) 166p	15. SECURITY CLASS. (of this report): Unclassified; 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document is approved for public release and sale; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

----

18. SUPPLEMENTARY NOTES

----

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

1: abstract programming, domain-independent interactive system, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process information.

2: computer terminals, interactive message service, Military Message Experiment, nonprofessional computer users, SIGMA, terminal-based message service

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes the research performed by USC/Information Sciences Institute from July 1, 1977, to September 30, 1978. The research is aimed at applying computer science and technology to areas of high DoD/military impact. The ISI program consists of thirteen research areas: Specification Acquisition From Experts - study of acquiring and using program knowledge for making informal program specifications more precise; Military Message Experiment - development of a user-oriented message service for large-scale

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407952

19. KEY WORDS (continued)

3: abstract data type, abstraction and representation, Alphard, Euclid, interactive theorem proving, Pascal, program correctness, program verification, rewrite rules, software specification, verification condition.
4: computer network, digital voice communication, network conferencing packet-switched networks, secure voice transmission, signal processing, speech processing, vocoding.
5: command and control, computer graphics, high-level graphics language, on-line map display.
6: Ada, Autopsy, CMS-2, DOD-1, program conversion, program equivalence, program translation, source-to-source transformation.
7: emulators, ISPS, microprogramming, MLP-900, multimicroprocessor emulation, National Software Works, program development tools, QM-1, QPRIM.
8: access control, computer security, encapsulation, error analysis, error-driven evaluation, error types, evaluation methods, protection mechanisms.
9: communication protocols, cooperation between decentralized processes, DSN modeling, packet radio network, position-location, rigidity, sensor networks.
10: computer mail, gateways, interconnection, internetwork protocol, networks, protocol design, protocols, protocol verification, type-of-service.
11: command and control, digital voice communication, graphic input device for terminal, multimedia communications, portable terminal, radio-coupled terminal.
12. ARPANET, naive computer users, TENEX, user assistance, user documentation.
13: ARPANET interface, computer network, FPS AP-120B, KA/KI, KL1090T, KL2040, PDP-11/45, resource allocation, TENEX, timesharing, TOPS-20.

20. ABSTRACT (continued)

military requirements; Program Verification - logical proof of program validity; Network Secure Communication - work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; Command and Control Graphics - development of a device-independent graphic system and graphics-oriented command and control applications programs; Autopsy - research program on source-to-source program translation combining automatic techniques with an interactive system to provide the human manager complete control over the translation process; Programming Research Instrument - development of a major time-shared microprogramming facility with an extension for emulation of microprocessors; Protection Analysis - methods of assessing the viability of security mechanisms of operating systems; Distributed Sensor Networks - developing algorithms and communication protocols to support the operation of geographically distributed sensors; Internetwork Concepts - effort exploring aspects of protocols for the interconnection of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; Packet Radio Terminal System Evaluation - work intended to result in a demonstration level portable terminal to test and evaluate various solutions to the issues raised by extreme portability in the packet radio environment; User-Dedicated Resource - assistance to and documentation for ARPANET users worldwide; and ARPANET TENEX Service - operation of TENEX service and continuing development of advanced support equipment.

ARPA ORDER NO. 2223

ISI/SR-79-14

1978
Annual Technical Report

July 1977 - September 1978

A Research Program in Computer Technology

Prepared for the

Defense Advanced Research Projects Agency

Effective date of contract: **17 May 1972**

Contract expiration date: **30 September 1978**

Principal Investigator and Director: **Keith W. Uncapher**
~~(213) 822-1511~~

Deputy Director: **Thomas O. Ellis**
(213) 822-1511

This research is supported by the Defense Advanced Research Projects Agency under Contract NO. DAHC15 72 C 0308, ARPA Order NO. 2223.

UNIVERSITY OF SOUTHERN CALIFORNIA *INFORMATION SCIENCES INSTITUTE* ISI

4676 Admiralty Way Marina del Rey California 90291

RESEARCH AND ADMINISTRATIVE SUPPORT

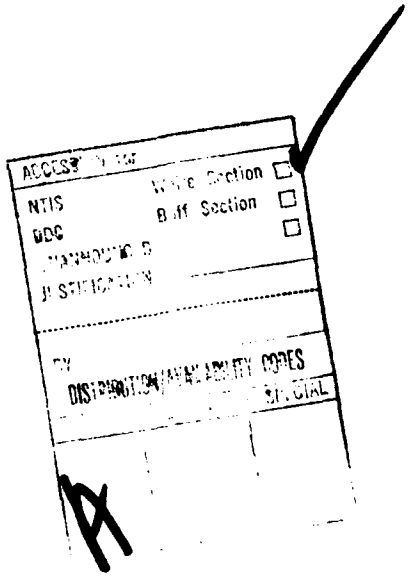
Institute Administration:
Robert Blechen
Judy Gustafson
Gina Maschmeier
Patricia Sefton
Pam Wilson

Librarian:
David Van de Streek

Publications Group:
Nancy Bryan
G. Nelson Lucas
Jim Melancon

Secretaries to Directors:
Patricia A. Craig
Arva Morgan

CONTENTS



SUMMARY

This report summarizes the research performed by USC/Information Sciences Institute from July 1, 1977, to September 30, 1978. The research is aimed at applying computer science and technology to areas of high DoD/military impact.

The ISI program consists of thirteen research areas: *Specification Acquisition From Experts* -- study of acquiring and using program knowledge for making informal program specifications more precise; *Military Message Experiment* -- development of an experimental user-oriented message service for potential large-scale military use; *Program Verification* -- logical proof of program validity; *Network Secure Communication* -- work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; *Command and Control Graphics* -- development of a device-independent graphic system and graphics-oriented command and control applications programs; *Autopsy* -- research program on source-to-source program translation combining automatic techniques with an interactive system to provide the human manager complete control over the translation process; *Programming Research Instrument* -- development of a major time-shared microprogramming facility with an extension for emulation of microprocessors; *Protection Analysis* -- methods of assessing the viability of security mechanisms of operating systems; *Distributed Sensor Networks* -- developing algorithms and communication protocols to support the operation of geographically distributed sensors; *Internetwork Concepts* -- exploring aspects of protocols for the interconnection of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; *Packet Radio Terminal System Evaluation* - work intended to result in a demonstration level portable terminal to test and evaluate various solutions to the issues raised by extreme portability in the packet radio environment; *User-Dedicated Resource* - assistance to and documentation for ARPANET users worldwide; and *ARPANET TENEX Service* - operation of TENEX service and continuing development of advanced support equipment.

EXECUTIVE OVERVIEW

The Information Sciences Institute (ISI), a research unit of the University of Southern California, performs research in the field of computer and communications sciences with an emphasis on systems and applications.

This document reports the progress and results of the projects at ISI supported by ARPA-IPTO. The significance and relevance of these programs are enhanced by the special character of ISI, which is an open, nonproprietary, university-based research center consisting of a staff of full-time professional computer scientists with graduate student support. ISI maintains strong active ties and direct involvement with the military services in identifying key R&D requirements and focusing the research areas.

The activities of ISI's major areas of research and associated support projects are summarized briefly below. Some of the research projects reported in this document are discrete activities in themselves; most can be seen as parts of a larger theme. For example, Program Verification, Specification Acquisition, Autopsy, and the Programming Research Instrument projects should be considered as individual parts of an overall research effort in programming methodology and quality software; the Military Message Experiment, Network Secure Communication, Distributed Sensor Networks, Internetwork Concepts, Command and Control Graphics, and Packet Radio Terminal are linked elements of a major investigation into man-machine and network communications technology. ARPANET TENEX Service and User-Dedicated Resource are projects whose major role is service to ARPANET users. This mutual reinforcement among the various projects at ISI contributes largely to the productivity of the Institute's research activities.

Specification Acquisition From Experts. It has been widely recognized that the road to improved software development begins with better software specification. Such specification must be formal, and many researchers are attempting to develop better languages for specifying software. But no matter how "good" such specification languages become, they will be difficult to use precisely because they are formal and--like any formal structure--require great care to build. During the last three years, the SAFE project has been exploring a different approach to simplifying software specification. Rather than developing a "better" specification language, we have developed a prototype system that acts as an interface between a software specifier and the formal specification that must be created. This interface (system) embodies our knowledge of what constitutes good formal software specifications, and it attempts to restate the user's *informal* software specification in

the required formalism. For the last few years we have been building a prototype system that embodies considerable knowledge of how people describe programs and what makes such programs well formed. This special knowledge provides a powerful basis for disambiguating informal program specifications. Our prototype system has understood and correctly formalized several real-world, albeit simplified, specifications. We believe that these examples demonstrated the basic feasibility of our approach; they also highlighted several deficiencies, some of which were resolved this year and reported herein. The resolution of the rest constitutes the basis of our plans.

Military Message Experiment. In 1975 ARPA, the Navy, and CINCPAC agreed to an operational test of an experimental message service, to be run at CINCPAC staff headquarters, Camp Smith, Oahu. In February 1977, ISI's SIGMA system was selected in a competitive evaluation as the message service best suited for the experiment. The Military Message Experiment (MME) is being conducted at CINCPAC headquarters, Oahu, to evaluate the utility of an interactive message service in a major military command. *SIGMA is a state-of-the-art system* developed by ISI expressly to meet the functional needs of the target community and the specialized requirements imposed by the experiment. It is a secure, on-line interactive writer-to-reader message service for the military community. It provides interactive assistance for formal messages from the initial draft preparation through coordination, transmission and distribution. SIGMA is primarily designed to handle AUTODIN messages, but it also supports internal record and non-record traffic. In order to keep storage requirements reasonable, messages are kept in a central data base and are shared by all users. Users are provided personal Folders to manage their message traffic. Entries in these Folders show an abstract of information from and are pointers to the real messages. Selectors are boolean filters which operate on the information in the entries and allow the user to restrict the messages he works on to one of interest. Both messages and Folders may be annotated with comments. SIGMA backs up the local two-dimensional editing in the MME terminal with powerful global editing features. Text may be stored for re-use in named text objects. In addition, a number of special operations are tailored to meet the message handling procedures at CINCPAC, both for the distribution of incoming messages and the coordination and release of outgoing messages. Though it is early in the experiment, *SIGMA is* already having a significant impact on the style of message processing by its users.

Program Verification. The goal of program verification research at ISI is to develop an effective program verification system for proving that computer programs are consistent with precisely stated specifications of what the programs are intended to do. The system is expected to replace significant parts of testing in current software development, and will also provide important tools for developing and judging the success of new programming language designs, new programming

methodologies, and new detailed specification techniques. The style of specifications and the methods of program construction, especially from isolatable program components, are all important influences in making verification feasible. Already running at ISI is an initial, experimental version of an interactive program verification system which accepts specifications of data abstractions and programs in a formal specification language that can be dynamically extended to applicable domains. The system has verified numerous example programs. Important progress has been made in the following areas: improved user environment and interface to the verifier, algebraic approach to data abstractions including their verification by a natural deduction theorem prover emphasizing rewrite rules, and experiments showing that methods and tools do apply to real, intermediate-sized, moderately complex software components. The eventual impact will be an increase in the quality of software.

Network Secure Communication. The major objective of ARPA's Network Secure Communication project is to develop secure, high-quality, low-bandwidth, real-time, two-way digital voice communication over packet-switched computer communication networks. This kind of communication is a very high priority military goal for all levels of command and control activities. ISI's role in this effort is to develop efficient user-oriented systems for digital voice communications, primarily over packet-switched networks such as the ARPANET. The ISI NSC project is working on network voice protocols, digital voice conferencing systems, voice-oriented network host operating systems, real-time signal processing, hardware development, and future integration of voice, graphics, and text into a powerful packet-based command, control, and communications system. During the past year the ISI NSC project has taken a series of measurements designed to increase our understanding of packet voice behavior on the ARPANET, has been involved in the specification and development of new types of network interfaces for packet voice systems, and has begun the development of a second-generation Network Voice Protocol.

Command and Control Graphics. In cooperation with the Navy, ARPA/IPTO is preparing a command and control testbed at Naval Ocean Systems Center to explore and demonstrate the use of advanced computer techniques in military applications. There is a recognized need for high-quality graphics input and output in this environment to accommodate effective data presentation and to support high-quality user interaction with command and control programs and data bases. In support of the testbed, the ISI Command and Control Graphics project is developing a graphics system that homogeneously supports graphics terminals of widely varying capability, potentially interacting with one or more different programs running on separate computers connected via a command and control communications network. The project is also developing advanced applications for the testbed that fully exploit the graphics medium.

Autopsy. The purpose of the Autopsy project is to build a semiautomatic program translation system for converting programs in old programming languages into a new programming language. The system being developed will provide a series of tools for employment under the user's direction. One of these will be an automatic translator, though it will probably not be able to translate every combination of constructs in the old language, or at least not into acceptably efficient new code. Consequently, the user will have the option to guide some parts of the translation more carefully. A history of the translation will be produced by the system to document the relationship between the old and the new code. The initial focus of the project is translating CMS-2M programs into Ada, DoD's new programming language for embedded computer systems.

Programming Research Instrument. The PRIM system is a multi-user, interactive high-speed emulation facility for the exploration of computer architectures and development and debugging of code for small- and medium-scale computers. It is based on the MLP-900, which is attached to a PDP-10 running TENEX. Current work extends the basic PRIM system in two directions. First, the system is being reimplemented with a QM-1 as the emulation engine in place of the MLP-900. When complete, the QM-1-based system (QPRIM) will be operational at both RADC and NOSC, where QM-1's will exist and be attached to DECSystem 20's running TOPS-20. The other major extension is the modeling and emulation of closely coupled sets of processors in order to support architecture studies for forthcoming multiprocessor systems, predominantly those being designed around microprocessors.

Protection Analysis. This project, which terminated in 1978, had as its goal the development of effective tools and techniques for detecting operating system protection errors (errors that allow the security of the system to be compromised). The approach was an empirical identification and analysis of error types and a development of effective search strategies to detect various types of errors.

Distributed Sensor Networks. The Distributed Sensor Networks project explores the problems arising from the need to integrate the technologies of computer communication networks, sensors and artificial intelligence. Our work consists of three major efforts. First we have been developing a "sensor-independent" DSN model whereby the problems of communication and artificial intelligence may be encapsulated independently of the details of any specific sensor. Second, we have been developing a simulation model of DSN and studying issues in real-time high-level protocols for DSN communications. Finally, we have developed extensive mathematical and algorithmic tools to solve the position-location problem and have implemented algorithms to study the practical aspects of the problem.

Internetwork Concepts. The Internetwork Concepts project explores four aspects of protocols for the interconnection of computer communication networks: (1) the design, and in some cases prototype implementation, of internetwork applications, (2) the design of internetwork host and gateway protocols, (3) the formal analysis of the host level protocol Transmission Control Protocol (TCP), and (4) general research on communication concepts for internetwork environments. The project aims to provide for appropriate and effective designs for the primary user service applications (i.e., Mail, Telnet, and File Transfer Protocol) in the internetwork environment, based on a set of host and gateway level protocols that provide the full range of service characteristics appropriate to a wide range of applications (e.g., speech, graphics, text), and that have been specified and analyzed to ensure their correct operation.

Packet Radio Terminal System Evaluation. ARPA is currently developing a Packet Radio Network (PRN) that will provide a wideband data communication capability much like the ARPANET but with the added dimensions of mobility and dynamic configurability. As this concept gains acceptance in the military services, fundamental choices will need to be made about mobile PR terminals for use with the system. This study, begun in June 1978, addresses the terminal characteristics desirable at the user level through the lower level protocol design decisions, the kind of interfaces required to support them, and the impact of such issues on terminal designs and other portions of the system. The work is intended to result in a demonstration level portable terminal to test and evaluate various solutions to the issues raised by extreme portability in the packet radio environment.

User-Dedicated Resource. This project was begun in order to provide encouragement and help to new ARPANET users who faced an initial barrier because of their lack of previous on-line experience, the rapidity of system modification, and the insufficiency of introductory documentation. New ARPANET users are contacted as soon as their accounts are installed on the ISI machines; they are interfaced to the available network facilities by means of three levels of appropriate documentation. Both short-term solutions (answering questions, solving individual problems) and long-term solutions (guiding users in solving common problems, making available better procedures and documentation) are routinely provided. Introductory primers and manuals have also been written exclusively for new users.

ARPANET TENEX Service. ISI is supporting, operating, and maintaining three TENEX and two TOPS-20 systems at ISI on a schedule of 161 hours per week each, in order both to provide TENEX service to ARPA and to support its research projects via the facilities at ISI. ISI operates two TENEX and TOPS-20 systems at a computer center that is part of the command and control testbed at the Naval Ocean Systems Center, San Diego, California. As part of the Military Message Experiment,

ISI also operates one TENEX system at Camp Smith, Oahu, on a 161 hour per week basis. The Institute provides a 24-hour availability of TENEX systems, maintenance, and operators, continued development/improvement support, support of the XGP at IPTO, as well as ARPA NLS user support and minimal NLS software support. Through this support we have achieved increased long-term up-time, faster repair and improved preventive maintenance, and economy of scale in operation.

1. SPECIFICATION ACQUISITION FROM EXPERTS

Research Staff:
Robert Balzer
Neil Goldman
Lee Erman

Research Assistant:
Chuck Williams
Mark James

Support Staff:
Joan C. Nosanov

INTRODUCTION

It has been widely recognized that the road to improved software development begins with better software specification. Such specification must be formal, and many researchers are attempting to develop better languages for specifying software. But no matter how "good" such specification languages become, they will be difficult to use precisely because they are formal and--like any formal structure--require great care to build.

During the last three years, the SAFE project has been exploring a different approach to simplifying software specification. Rather than developing a "better" specification language, we have developed a prototype system that acts as an interface between a software specifier and the formal specification that must be created. This interface (system) embodies our knowledge of what constitutes good formal software specifications, and it attempts to restate the user's *informal* software specification in the required formalism.

We recognize that this is a radical approach to the problem, but we believe it is essential to recognize that formal mechanisms are foreign to man's nature. We can be taught--with considerable pain, difficulty, and expense--to use them, but we always revert to informal structures wherever possible. Even mathematical proofs are only partially formalized arguments that a given line of reasoning is correct.

Formal software specifications must be produced, but that does not imply that people *should* create them. Rather, people should continue to write informal specifications (such as existing B-5 military software specifications) and use a computer-based tool to produce the formal software specifications. Such a division of effort between man and machine seems much more appropriate, with the computer system responsible for reformulating the user's informal specification in the particular form required, thus maintaining consistency, avoiding ambiguity, and ensuring completeness.

This is basically the standard language translation task; it is not difficult if the difference between the input and output languages is small and the input is understood. To eliminate the first of these problems we have specifically designed our formal specification language to minimize the differences between it and the informal input. Thus the key issue is understanding the input.

This difficulty in understanding the input specification has limited previous efforts to the use of formal specification languages while precluding informal specification--a limitation that shifted the understanding burden completely to the human specifier. We believe this extreme position is unwarranted and imposes an artificial barrier between computers and potential users. In a suitably constrained environment, much of the understanding of an informal input can be handled by a computer system which then undertakes the responsibility of re-expressing it in the required formalism.

How can such understanding be automated? First, we must define the nature of informal languages. The primary distinction between formal and informal language is that in the latter information is suppressed, yielding partial descriptions and constructs rather than complete ones. These partial descriptions and constructs result in ambiguity that can be resolved only by context.

But what information is suppressed? It is far from random. The originator of the informal communication suppresses information that he believes the recipient can correctly fill in and may temporarily suppress details not central to the main topic being communicated. Thus informal languages are effective because they allow both the originator and the recipient to focus on the relevant issues and to suppress details that can be correctly inferred and that, if present, would detract from or blur the issues being highlighted.

Therefore, to automate the understanding of informal languages one must disambiguate partial descriptions and constructs by supplying the information suppressed because the originator felt it to be obvious from the context. Unfortunately, providing such disambiguation has been remarkably difficult because of the large variability in how context affects the disambiguation and the undetermined amount of unstated world knowledge that might bear upon the resolution.

On the other hand, considerable success can be gained by automating disambiguation in a highly constrained domain. Fortunately, programs are very highly constrained objects (one reason they are so hard to construct), and people seem to describe them in rather limited ways.

For the last few years we have been building a prototype system that embodies considerable knowledge of how people describe programs and what makes such programs well formed. This special knowledge provides a powerful basis for disambiguating informal program specifications. Our prototype system has understood and correctly formalized several real-world, albeit simplified, specifications. We believe that these examples demonstrated the basic feasibility of our approach; they also highlighted several deficiencies, some of which were resolved this year and reported herein. The resolution of the rest constitutes the basis of our plans.

ACCOMPLISHMENTS AND CURRENT STATUS

Over the last several years, we have developed the basic technology for computer comprehension of informal program specifications and for their translation into a precise operational formalism. This accomplishment has been demonstrated by applying the system to several simplified military systems whose informal specifications were extracted from existing MIL-SPEC 490-B5 Software Specification Manuals (see Appendix I).

This technology is based upon an extensive set of rules defining the allowable dynamic behavior of well-formed processes used to disambiguate the possible interpretations of constructs in the informal specification. Only those interpretations that lead to well-formed behavior are accepted (see Appendix II for a technical description of the SAFE system). Our experience to date indicates that these general rules (augmented with task-specific rules, that normally accompany the informal specification) provide sufficient disambiguation power with only minor user interaction.

LIMITATIONS

Naturally, these first results have been achieved in a highly restricted context. First, only small (approximately 10- to 15-sentence) specifications have been handled. These are far from practical military size, and our project faces a major task of scaling the current system to handle practical-sized specifications (this task is the central thrust of our FY79 and FY80 plans).

Second, only a subset of the necessary software constructs are understood by the existing system; for example, parallel processing, interrupts, systems with memory, and exception handling are omitted. Basically, the current system is limited to single-process systems operating in a static environment (one changed only by the program itself) without permanent memory (the environment is reset

between successive sets of inputs so that the processing of one does not affect any of the others). This is an important subset of systems and is clearly the simplest place to start.

Third, the very nature of informal languages means that there is no single way to specify a system; instead, a very broad range of alternatives must be accepted. The original system was quite limited in this regard, and increasing the system's robustness was the main thrust of this year's activity (described below).

Fourth, informal specification techniques are important for the entire range of military software systems. A separate informal-to-formal translation cannot be built for each system, or even for each application area; instead, a common system must be reused. Thus our system has been built to be independent of any particular application. So far, we have been quite successful in this regard by segregating the knowledge specific to an application into data tables used by the system. However, creating these data tables is itself quite laborious and error-prone; a major effort is required to apply the existing system to a new application.

Finally, a usable version of our system requires a user interface that accepts the informal specification, interacts with the user to solicit information to complete, correct, and/or disambiguate the specification, and explain the formal specification derived. We would like to use natural language for this interface, but neither our system nor the technology of natural language parsing is sufficiently mature at this point. Additionally, there are many others more qualified than ourselves to pursue the problem of natural language parsing. Therefore, we have chosen to avoid this issue for the time being by manually preparing the informal specification (accomplished by enclosing each noun phrase and verb phrase in parentheses). This enables us to concentrate our effort on the semantic issues of comprehending informal languages.

Both our current and proposed plans are focused on gradually removing these restrictions.

CURRENT ACTIVITY AND PROGRESS (FY78)

Our major activity this year was to extend the system's robustness to variability in the informal specifications. Specifically, we have created 25 perturbations of the examples already handled by the system. Each is a quite reasonable variation of the original specification, but each has been chosen so that the current system (as of the beginning of FY78) would not handle it. Thus each perturbation forced us to extend the system's capacity to handle input variability.

The following examples illustrate the types of perturbations chosen and the extensions of the system required to handle these perturbations (the original specifications are presented in Appendix I).

1. *In the message processing example (see Fig. I.1 of Appendix I) change sentence 7 to read "perform the action associated with the key" instead of "perform the action associated with the type of key".*

   Effect: The domain model is changed so that actions are associated with keys rather than with the type of the key. This causes the later statement of the association between a particular action (named action-0 by the system) and type-0 keys to be interpreted as an inference rule which says that if a key has the property of being a type-0 key, then its associated action is action-0.

   Original processing: Because explicit links existed between the key types and specific actions, the "perform" statement was translated into a case statement for the analysis of this specification so that it could be determined that assignments could occur during these actions and hence that the searching which initiates these actions is part of the assignment process.

   Difficulty: The explicit associations between actions and keys has become implicit (represented as an inference rule). Thus a set of alternatives is not explicitly available to gather together into a case statement. Notice that this difficulty is simply the change from explicit to implicit data and has nothing to do with the domain model shift from associations between actions and key types to keys themselves. This shift is adequately handled by a general "association" mechanism and would have been successfully handled by our unaugmented (FY78) system. The difficulty was caused by the introduction of implicit data necessary for determining the control structure of the specification.

   System extension: An ability to reason about the set of data derivable from an inference rule, similar to the capability existing within the Meta-evaluation phase, was added to the Planning phase.

2. *In the Link Scheduling example (see Fig. I.4 of Appendix I), in sentence 2, remove the predicate limiting the computation of a relative transmission time for a subscriber to only those situations in which a subscriber entry was found in the SOL.*

Effect: A control structure for the specification will be formulated in which the computation of the relative transmission time for a subscriber is performed in inappropriate situations (when no entry for that subscriber exists within the SOL).

Difficulty: This control structure causes an uncorrectable (via normal backtracking mechanism) error because the data required for the computation is not guaranteed to be available.

System extensions: Certain unanticipated "control flow" type errors are part of the informal nature of the specification and can be handled with the addition of specialized "fixer" experts. This is the first instance of handling unanticipated informal constructs. Previously the informal nature of a construct was recognized upon entry and an explicit decision mechanism was employed to investigate the alternatives. Here the lack of well-formedness of the specification leads to the identification of the cause (or possible causes) of the problem that are resolved by special purpose mechanisms.

3. *In the Narrative Message Analysis example (see Fig. I.7 in Appendix I) change "messages with an addressee matching a CGL entry" in sentence 7 to "message matching a CGL entry".*

Effect: The change causes the system to match messages, rather than addressees within the messages, against CGL entries (which are also addressees).

Difficulty: Such matches are inappropriate because a message could never match an addressee, but the system was unaware of this ill-formed structure.

System extension: The semantics of "compare" was strengthened so that it was known that type compatibility was a necessary (but not sufficient) condition for a match to occur, and the system was extended to use necessary and/or sufficient conditions in its determination of well-formedness. With this extension, the system was able to determine that the arguments to the comparison (which the system added to produce matches) must be type-compatible with each other so that a match is, in fact, a possible result. As in the previous example perturbation, this causes the system to dynamically recognize that the match specification was informal and that it be well-formed, the arguments must be type-compatible. The existing general association mechanism was capable of satisfying this requirement when invoked by a specialized "fixer"

expert. The situation is one in which a type coercion is needed, but the appropriate target type is a function not only of the relation and operand, but of another operand as well.

PLANS

The major effort for FY79 is to scale the SAFE system so that it can handle real unsimplified specifications of practical size. A specific example, the ARPANET Host-IMP Protocol, has been chosen because it is well-written, widely known, and representative of an important class of applications. FY79 will be spent preparing SAFE for specifications of this class and size (about 20 pages).

Specifically, we will start analyzing the Host-IMP Protocol specification in detail to determine what capabilities are required to understand this class of specifications. General knowledge about this class of system will be identified (for example, that a protocol is a set of rules for exchanging information between two or more processes and that transmission errors can occur), and methods of incorporating it into the system devised. Deficiencies in the current system and/or the protocol specification will be identified; a plan will then be generated and implementation begun on overcoming these difficulties.

Orthogonal to these issues specific to the class of protocol problems is a set of issues concerned merely with handling larger specifications. These issues include limiting the size of the knowledge base created and managing its contents; identifying the points of search within the system and applying more powerful heuristics to prevent combinatorial explosion; and determining how to process the specification a part at a time, with limited interaction between the parts, rather than considering it as an unpartitioned whole.

Also, we have particular notions about how specifications should be written, which center on the issues of separating functionality from representation and optimization, and of providing abstract yet process-oriented specifications. These notions must be explicated as a set of standards for SAFE specifications. Then the Host-IMP protocol must be rewritten to conform to these standards. In particular, it is currently filled with representation issues that must be removed.

Because we measure progress and test system capabilities through particular examples, we will generate a few simplified versions of the rewritten specifications that allow us to consider these issues individually and work on them as the implementation proceeds. We also plan to gain experience with a somewhat larger specification as a prelude to scaling up our system to handle practical-sized systems. We have chosen an example approximately three times larger (about 45

sentences) than any we have previously handled. It describes a military electronics switchboard in which an operator uses a keyboard to direct and control the interconnection of subscripts via various types of circuits.

In FY80, we will actually begin working on the Host-IMP Protocol. The rewritten version, created the previous year to conform to SAFE standards for specifications, will be converted to a formal specification for the Host machine by mid year. In the remainder of the year, we will create several perturbations of this specification to test the system's robustness to variability of large specifications.

Furthermore, we will provide a rudimentary capability to exercise the formal specification. This will enable a user to watch the formal specification operate on selected data to ensure that the formal specification produced by the system matches the user's intent. Such a capability is important because just as it is difficult for a user to write a formal specification directly, so is it difficult for him to directly (i.e., by inspection) understand the formal specification produced by the system. Instead, by observing the behavior of the formal specification on appropriate cases, the correspondence between intent and result can be ascertained. Together with a paraphrase capability (to be added later as part of the user interface), this should provide a quite complete capability for user understanding of a formal specification.

Such a capability to exercise a formal specification is important for a second reason. In addition to providing information concerning the correspondence between the formal specification and the user's intent, it also provides an assessment of whether the specification meets the user's requirements. Short of producing a formal statement of these requirements (with all the attendant difficulties of such formalization) and verifying that the formal specification is consistent with these formal requirements, the only recourse is to make such an assessment by appropriately testing (i.e., observing the behavior of) the specification. Rome Air Development Center (RADC) is currently considering such an effort (testing specifications) as an outgrowth of the SAFE project.

APPENDIX 1

This section presents three examples successfully handled by the SAFE prototype system. The examples were extracted from actual natural language specification manuals, and the results illustrate the power of the system's context mechanisms. However, our system is only a prototype and, as such, it is far from complete. New examples currently expose new problems that are resolved by adding new capabilities to the system. Therefore, until some measure of closure is obtained, it should not be assumed that the prototype will correctly process new examples of the same "complexity" as earlier examples. Our goal is to add each new capability in as general a form as possible so that when it is used in new examples it will function correctly. In this way we expect to "grow" the system as more complex and varied examples are tried.

For each of the examples, we present three figures: the actual parenthesized version of the informal input currently used by the system (to avoid syntactic parsing problems), a manually marked version, which indicates some of the informalities to be resolved by the system, and a stylized version of the formal output program produced by the system. These programs are expressed in (and run on) our own language (AP2), which uses a relational data base as the repository for all data manipulated by the programs.

The first example is a system that automatically distributes messages to offices on the basis of a keyword search of the text of the message. Figure 1.1 gives the informal natural language description. Figure 1.2 indicates some of the imprecisions contained in this example that must be resolved to obtain the system's formalization of this specification as an operational program (Fig. 1.3).

To give some measure of the amount of imprecision in this example and, therefore, the amount of aid provided by the system, we have compiled the following statistics:

Number of missing operands	=	18
Number of incomplete references	=	22
Number of implicit type conversions	=	9
Number of terminology changes	=	3
Number of refinements or elaborations	=	2
Number of implicit sequencing decisions	=	7

ACTUAL INPUT FOR MESSAGE PROCESSING EXAMPLE

*((MESSAGES ((RECEIVED) FROM (THE "AUTODIN-ASC"))) (ARE PROCESSED) FOR (AUTOMATIC DISTRIBUTION ASSIGNMENT))

*((THE MESSAGE) (IS DISTRIBUTED) TO (EACH ((ASSIGNED)) OFFICE))

*((THE NUMBER OF (COPIES OF (A MESSAGE) ((DISTRIBUTED) TO (AN OFFICE)))) (IS) (A FUNCTION OF (WHETHER ((THE OFFICE) (IS ASSIGNED) FOR (("ACTION") OR ("INFORMATION"))))))

*((THE RULES FOR ((EDITING) (MESSAGES))) (ARE) (: ((REPLACE) (ALL LINE-FEEDS) WITH (SPACES)) ((SAVE) (ONLY (ALPHANUMERIC CHARACTERS) AND (SPACES))) ((ELIMINATE) (ALL REDUNDANT SPACES))))

*(((TO EDIT) (THE TEXT PORTION OF (THE MESSAGE))) (IS) (NECESSARY))

*(THEN (THE MESSAGE) (IS SEARCHED) FOR (ALL KEYS))

*(WHEN ((A KEY) (IS LOCATED) IN (A MESSAGE)) ((PERFORM) (THE ACTION ((ASSOCIATED) WITH (THAT TYPE OF (KEY))))))

*((THE ACTION FOR (TYPE-0 KEYS)) (IS) (: (IF ((NO OFFICE) (HAS BEEN ASSIGNED) TO (THE MESSAGE) FOR ("ACTION")) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS ASSIGNED) TO (THE MESSAGE) FOR ("ACTION"))) (IF ((THERE IS) ALREADY (AN "ACTION" OFFICE FOR (THE MESSAGE))) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS TREATED) AS (AN "INFORMATION" OFFICE))) (((LABEL OFFS1 (ALL "INFORMATION" OFFICES FROM (THE KEY))) (ARE ASSIGNED) TO (THE MESSAGE)) IF ((REF OFFS1 THEY) (HAVE (NOT) (ALREADY) BEEN ASSIGNED) FOR (("ACTION") OR ("INFORMATION"))))))

*((THE ACTION FOR (TYPE-1 KEYS)) (IS) (: (IF ((THE KEY) (IS) (THE FIRST TYPE-1 KEY ((FOUND) IN (THE MESSAGE)))) THEN ((THE KEY) (IS USED) TO ((DETERMINE) (THE "ACTION" OFFICE)))) (OTHERWISE (THE KEY) (IS USED) TO ((DETERMINE) (ONLY "INFORMATION" OFFICES)))))

Figure 1.1

SPECIFICATION DEFICIENCIES OF
MESSAGE PROCESSING EXAMPLE
(BY CONVENTIONAL PROGRAMMING STANDARDS)

* MESSAGES RECEIVED FROM THE AUTODIN-ASC ARE PROCESSED ~~FOR~~ AUTOMATIC ~~DISTRIBUTION~~ ASSIGNMENT.

* THE MESSAGE IS DISTRIBUTED TO EACH ASSIGNED OFFICE.

* THE NUMBER OF COPIES OF A MESSAGE DISTRIBUTED TO AN OFFICE IS A FUNCTION OF WHETHER THE OFFICE IS ASSIGNED FOR ACTION OR INFORMATION.

* THE ~~RULES~~ FOR EDITING MESSAGES ARE (1) REPLACE ALL LINE-FEEDS WITH SPACES (2) SAVE ONLY ALPHANUMERIC CHARACTERS AND SPACES AND THEN (3) ELIMINATE ALL REDUNDANT SPACES.

* IT IS NECESSARY TO EDIT THE TEXT PORTION OF THE MESSAGE.

* THE MESSAGE IS THEN SEARCHED FOR ALL KEYS.

* WHEN A KEY IS LOCATED IN A MESSAGE, PERFORM THE ACTION ASSOCIATED WITH THAT TYPE OF KEY.

* THE ACTION FOR TYPE-0 KEYS IS: IF NO ACTION OFFICE HAS BEEN ASSIGNED TO THE MESSAGE, THE ACTION OFFICE ~~FROM~~ THE KEY IS ASSIGNED TO THE MESSAGE FOR ACTION. IF THERE IS ALREADY AN ACTION OFFICE FOR THE MESSAGE, THE ACTION OFFICE FROM THE KEY IS TREATED AS AN INFORMATION OFFICE. ALL INFORMATION OFFICES FROM THE KEY ARE ASSIGNED TO THE MESSAGE IF THEY HAVE NOT ALREADY BEEN ASSIGNED FOR ACTION OR INFORMATION.

* THE ACTION FOR TYPE-1 IS: IF THE KEY IS THE FIRST TYPE-1 KEY FOUND IN THE MESSAGE THEN THE KEY IS USED TO DETERMINE THE ACTION OFFICE. OTHERWISE THE KEY IS USED TO DETERMINE ONLY INFORMATION OFFICES.

Figure 1.2

PROGRAM CREATED BY PROTOTYPE SYSTEM

```
(WHENEVER (receive message FROM autodin-asc BY safe)
 DO(edit text OF message)
  (search text OF message FOR (CREATE THE SET OF keys))
  (distribute-process#1 message))

(distribute-process#1 (message)
 (FOR ALL (offices assigned TO message FOR ANYTHING)
  (distribute-process#2 message office)))

(distribute-process#2 (message office)
 (DO (function#1 (BOOLEAN (assigned office TO message FOR action))
  (BOOLEAN (assigned office TO message FOR information)))
 TIMES (distribute A copy WHICH IS A copy OF message AND located
  AT safe FROM safe TO location OF office)))

(edit (text)
 (FOR ALL line-feeds IN text
  (replace line-feed IN text BY (CREATE SET OF spaces)))
 (keep (union (CREATE THE SET OF alphanumeric characters IN text)
    (CREATE THE SET OF spaces IN text))
  FROM text)
 (FOR ALL spaces IN text AND redundant IN text
  (remove space FROM text)))

(WHENEVER (locate A key IN text OF message AT POSITION ANYTHING)
 DO(CASE (type OF key)
  (type-0 (type-0-action message key))
   (type-1 (type-1-action message key))))

(type-0-action (message key)
 (IF (NOT (EXISTS action office FOR message))
  THEN (assign THE action office#1 FOR key
   TO message FOR action)
 ELSE (treat action office#2 FOR key
   AS information office#2 FOR key
  IN (IF (NOT (assigned office#2 TO message
      FOR action OR information))
   THEN (assign office#2 TO message FOR information))))
 (FOR ALL (office#3 assigned TO key FOR information)
  (IF (NOT (assigned office#3 TO message
   FOR action OR information))
   THEN (assign office#3 TO message FOR information))))
```

```
(type-1-action (message key)
 (IF key = (key#1 WHICH IS (SEARCH HISTORY FOR FIRST
  (locate type-1 key#1 IN text OF message AT position ANY)))
   THEN (determine THE action office FOR message
   BY (type-0-action message key))
  ELSE (determine ONLY THE information office FOR message
   BY (IF (EXISTS action office FOR message)
      THEN (treat action office#1 FOR key
        AS information office#1 FOR key
        IN (IF (NOT (assigned office #1 TO message
            FOR action OR information))
          THEN (assign office#1 TO message FOR information))))
      (FOR ALL office#2 assigned TO key FOR information)
        (IF (NOT (assigned office#2 TO message
          FOR action OR information))
          THEN (assign office#2 TO message
            FOR information))))))
```

Figure 1.3

To illustrate how context is used to complete the partial descriptions in the example, we consider a few cases:

1. *Partial sequencing.* Distribution is never explicitly invoked in the informal specification. However, the first sentence indicates that Assignment is performed to enable the Distribution. Hence, Distribution should be explicitly invoked after Assignment.

2. *Missing operand.* Sentence 2 indicates that the message should be distributed to certain offices--those that are "assigned." But, as can be determined from other usages in the informal specifications, offices can be "assigned" to either messages or keys. This missing operand can be resolved by remembering that Assignment was performed to enable Distribution. Hence, Distribution must use some result of the assignment process. Assignment, from the last two input sentences, assigns offices to the current message. Hence, Distribution must use the offices assigned to that message.

3. *Incomplete reference.* Sentence 4 says to replace all line feeds with spaces. First, replace requires a third operand, some set in which the replacement will occur. Context indicates that this missing operand should be the text of the message parameter of Edit. Second, the use of a plural in the operand of an action which expects a singular operand, indicates an implicit loop. Hence, we have "for all line feeds, replace the line feed by a space in the text of the message." Now, which line feeds are we concerned with? Only those in the text of the message because they are the only ones which can be replaced. Hence, completing the partial reference, we have "for all line feeds in the text of the message, replace the line feed by a space in the text of the message."

It should be noted that of the approximately 61 decisions which had to be made for this example, all but one were resolved correctly by the prototype system. The message it distributed is the edited one (with all punctuation removed) rather than the original unedited one. The cause of the error is that the system does not understand the difference between an object being changed and its participating in relations with other objects; therefore, it has no concept of the original state of an object and hence does not consider this as a possible completion of any partial reference.

This capability can clearly be added to the system, but the important point is that interpretation errors will occur, just as they do when human intermediaries are used to produce the formal specification. It is therefore essential to provide extensive feedback and assumption-testing facilities so that such errors, when made, can be discovered and corrected by the user.

The second example is from a system for scheduling a satellite communication channel by multiplexing it among several users (subscribers). It specifies the component of the system which receives a schedule (SOL) from the controller of the satellite channel and extracts from it the portion of the next transmission cycle which has been reserved for a particular subscriber and those portions available to any user (RATS). This information is placed in a transmission schedule used by another component to actually utilize the channel during the allowed periods. Figure 1.4 gives the informal natural language description. Figure 1.5 indicates some of the imprecisions contained in this example that must be resolved to obtain the system's formalization of the specification as an operational program (Fig. 1.6). In addition to the process description of Fig. 1.4, we have assumed that the formulas referenced and a structural description of the objects of the domain have been separately specified.

The relevant portions of these specifications are that the SOL is an ordered set of subscriber and RATS entries. Each subscriber entry has subscriber identifier and transmission length fields, while a RATS entry has only the latter. The transmission schedule is a set of entries, each of which is composed of an absolute transmission time and a transmission length. One of these entries is the primary entry of the transmission schedule. Finally, formulas 1 and 2 both take an SOL entry as input and produce, respectively, a relative and an absolute transmission time.

ACTUAL INPUT FOR LINK SCHEDULING EXAMPLE

```
((THE SOL)
 (IS SEARCHED)
 FOR
 (AN ENTRY FOR (THE SUBSCRIBER)))

(IF ((ONE)
     (IS FOUND))
    ((THE SUBSCRIBER'S (RELATIVE TRANSMISSION TIME))
     (IS COMPUTED) ACCORDING-TO ("FORMULA-1")))

((THE SUBSCRIBER'S (CLOCK TRANSMISSION TIME))
 (IS COMPUTED) ACCORDING-TO ("FORMULA-2")))

WHEN ((THE (TRANSMISSION TIME))
      (HAS BEEN COMPUTED))
     ((IT)
      (IS INSERTED)
      AS (THE (PRIMARY ENTRY))
      IN (A (TRANSMISSION SCHEDULE))))

FOR (EACH RATS ENTRY)
    (PERFORM)
    (: ((THE RATS'S (RELATIVE TRANSMISSION TIME))
        (IS COMPUTED) ACCORDING-TO ("FORMULA-1"))
       ((THE RATS'S (CLOCK TRANSMISSION TIME))
        (IS COMPUTED) ACCORDING-TO ("FORMULA-2"))))

((THE RATS (TRANSMISSION TIMES))
 (ARE ENTERED)
 INTO (THE SCHEDULE))
```

Figure 1.4

SPECIFICATION DEFICIENCIES OF

LINK SCHEDULING EXAMPLE

* THE SOL IS SEARCHED FOR AN ENTRY ~~FOR~~ THE SUBSCRIBER.
* IF ONE IS FOUND, THE SUBSCRIBER'S RELATIVE TRANSMISSION TIME ~~IS COMPUTED ACCORDING TO FORMULA-1~~ and then
* THE SUBSCRIBER'S CLOCK TRANSMISSION TIME IS COMPUTED ACCORDING TO FORMULA-2.
* WHEN THE TRANSMISSION TIME HAS BEEN COMPUTED, IT IS INSERTED AS THE PRIMARY ENTRY IN A TRANSMISSION SCHEDULE.
* FOR EACH RATS ENTRY, THE RATS'S RELATIVE TRANSMISSION TIME IS COMPUTED ACCORDING TO FORMULA-1 AND THE RATS'S CLOCK TRANSMISSION TIME IS COMPUTED ACCORDING TO FORMULA-2.
* ~~THE~~ RATS TRANSMISSION TIMES ARE ENTERED INTO THE SCHEDULE.

Figure 1.5

PROGRAM CREATED BY PROTOTYPE SYSTEM

```
(build-transmission-schedule (sol subscriber)
   (CREATE transmission-schedule)
   (search sol FOR A subscriber-entry SUCH THAT
    sid OF subscriber EQUALS sid OF subscriber-entry)
   (IF (locate A subscriber-entry SUCH THAT
    sid OF subscriber EQUALS sid
      OF subscriber-entry IN sol)
    THEN
     (MAKE (RESULT-OF (FORMULA-1 subscriber-entry))
      BE THE relative-transmission-time OF subscriber)
     (MAKE (RESULT-OF (FORMULA-2 subscriber-entry))
      BE THE clock-transmission-time OF subscriber))
   (FOR ALL rats WHICH ARE IN sol
      DO (MAKE (RESULT-OF (formula-1 rats))
            BE THE relative-transmission-time OF rats)
         (MAKE (RESULT-OF (formula-2 rats))
            BE THE clock-transmission-time OF rats))
   (FOR ALL clock-transmission-time OF rats
      DO (MAKE clock-transmission-time BE THE
            transmission-time OF (CREATE transmission-entry))
          (ADD transmission-entry TO transmission-schedule)))

(WHENEVER (MAKE time BE THE clock-transmission-time
    OF subscriber)
      DO (MAKE time BE THE transmission-time
            OF (CREATE transmission-entry))
         (ADD transmission-entry TO transmission-schedule)
         (MAKE transmission-entry BE THE primary-entry
           OF transmission-schedule))
```

Figure 1.6

Using the same measures of imprecision as in the first example, we find that this example has about half as many imprecisions.

Number of missing operands	=	7
Number of incomplete references	=	12
Number of implicit type conversions	=	3
Number of terminology charges	=	0
Number of refinement or elaborations	=	0
Number of implicit sequencing decisions	=	4

The example is interesting as a test of the generality of the mechanisms that worked on the first example, and because of the new issues it raises. We will examine each of these to illustrate the range of capabilities added to the prototype to enable it to correctly understand this example and produce the operational program of Fig. 1.6.

1. *Scope of conditional.* In natural language communication the end of a conditional is almost never explicit. Instead, context must be used to determine whether subsequent statements are part of the conditional. In sentence 3 of the example, the input to formula 2 is the SOL entry found in the previous sentence. Thus, sentence 3 is really part of the conditional statement.

2. *Implicit formation of relations.* In sentence 2, the relative transmission time produced by formula 1 is supposed to be associated with the subscriber. Since that association is not established elsewhere, it is implicitly being established here. Hence this passive construct must be treated as an active one.

3. *Implicit creation of outputs.* In a similar fashion, various sentences establish associations with a transmission schedule (the output of this example) but an instance of one is never explicitly created. Such usage indicated that an implicit creation of the output is required.

4. *Expectation failure.* In addition to process and structural statements, a specification normally contains expectations about the state of the computation at some point which provide context for people to explain why something is being done or some properties of its result. They also provide some redundancy against which an understanding of the specification can be checked. In the example, one of these expectations (that all of the components of the entries of the output have been produced) fails, which indicates either a misunderstanding of the

specification or an inconsistency or incompleteness. In this case, both our example and the actual specification from which it was drawn are incomplete; they fail to describe how the length field of the entries of the transmission schedule are calculated from the inputs.

The third example is from the same satellite communication system as the previous example. This portion specifies the component of the system which determines whether the entire text of a message received over the satellite channel should be printed, or only its header. This determination is based on the type of message, for which several different cases are specified, and upon whether the message is "addressed" to the subscriber. This, in turn, is determined by seeing if the message contains an addressee which is in one set (the CGL) but not in another (the NP).

The parenthesized informal specification used as the actual input for the third example is shown in Fig. 1.7. Figure 1.8 shows some of the informal constructs contained in the input, and a stylized version of the formal operational abstract program produced by the prototype system is contained in Fig. 1.9.

ACTUAL INPUT FOR NARRATIVE MESSAGE ANALYSIS EXAMPLE

((The CGL) (is) (a list of (addressees))).

((The NP) (is) (a list of (addressees))).

((((Link messages)) ((first-run messages))
and ((normal mode) (rerun broadcast messages)))
(are screened) by ((comparing) (all addressees in
(the message)) with (the CGL)) in-order-to ((determine)
(whether ((the message) (is) ((addressed) to (the subscriber)).

(During ((screening)) (initially (compare) (an addressee)
with (the NP))).

(If ((the addressee) (matches)) ((continue) ((screening))
with (the next (message) addressee))).

((Messages with (an addressee ((matching) (a (CGL)
entry) (are) ((addressed) to (the subscriber))).

(Context: (((Link messages)) and ((first-run broadcast
messages)).

((Messages ((addressed) to (the subscriber))) (are noted)
so-that ((the message) (will be printed))).

((Messages (not ((addressed) to (the subscriber) (are
noted) so-that ((the (message heading)) (will be printed))).

((((First-run RS messages)) and ((first-run OLE messages)))
(are noted) so-that ((the message) (will be printed))).

(Not ((do screen) (((first-run RS messages)) and ((first-run
OLE messages)).

(((OLE messages)) (are output) on (the (utility punch))).

Figure 1.7

Specification Deficiencies of Narrative Message Analysis Example

The CGL is a list of addresses.

The NP is a list of addresses.

Link messages first-run messages and normal mode rerun broadcast messages are screened by comparing ~~all~~ each addresses in the message with the elements of the CGL in-order to determine whether the message is addressed to the subscriber.

During screening initially compare an an addressee in the message with the elements of the NP. then

If the addressee matches result continue screening with the next message addressee. in the otherwise

inference rule; Messages ~~with~~ which contain an addressee matching a CGL entry are addressed to the subscriber.

Context: Link messages and first-run broadcast messages.

Any Messages addressed to the subscriber ~~are~~ is noted with mark M1 so-that the message will be printed.

Any Messages not addressed to the subscriber ~~are~~ is noted with mark M2 so-that the message heading will be printed.

First-run RS messages and first-run OLE messages are noted with mark M1 so-that the message will be printed.

Do not screen except first-run RS messages and first-run OLE messages.

OLE messages are output on the utility punch.

PROGRAM CREATED BY PROTOTYPE SYSTEM

```
INFERENCE RULE:
        (message-addressee addressee IN message)
  AND    (match addressee addressee#1)
    WHERE (element addressee#1 cgl)
  IMPLIES
        (addressed message TO subscriber)
-------------------------------------------------------
Screen (message)
Loop1:
 FOR EACH  (message-addressee addressee#1
         IN message)
      DO
       FOR EACH  (element addressee#2 NP)
         DO  (compare addressee#1 TO addressee#2
            BY eq)
       IF  (match addressee#1 TO addressee#3 BY eq)
          WHERE  (element addressee#3 np)
         THEN  (continue loop1 with addressee#4)
          WHERE  (message-addressee addressee#4
                IN message)
            AND (successor addressee#4 OF addressee#1)
      FOR EACH  (element addressee#5 cgl)
       DO  (compare addressee#1 TO addressee#5 BY eq)
-------------------------------------------------------
Top-Level-Program (message subscriber utility-punch)
IF  (is message A first-run-message) OR (is message
   A normal-mode-rerun-broadcast-message)
 THEN IF (not (is message A first-run-rs-message)
       OR (is message A first-run-ole-message))
   THEN DETERMINE WHETHER (addressed message TO
                  subscriber)
        BY (screen message)
IF (addressed message TO subscriber)
  THEN (mark message WITH mark#1)
      FOR PURPOSE (print message)
  ELSE IF (not (addressed message TO subscriber)
      THEN (mark message WITH mark#2)
       FOR PURPOSE (print header)
      WHERE (message-header header OF message)
IF (is message A first-run-rs message) OR
   (is message A first-run-ole message)
   THEN (mark message WITH mark#1)
       FOR PURPOSE (print message)
IF (is message A ole-message)
   THEN (output message ON utility-punch)
```

Figure 1.9

In addition to the types of informality described previously, this example also contains the following:

1. *Discovered parameters.* Normally, references to objects within a procedure body are resolved by equating them to some known objects (a parameter, iteration variable, or variable determined by a data base pattern match) or by finding a close association between one of the known objects and the reference. When both these methods fail, the prototype system recognizes that it is unable to resolve the reference in the current context. It therefore assumes that the reference must be resolved in some larger context (i.e., the context existing when the procedure is called). It ensures this resolution by making the unresolved reference a parameter of the routine and requiring that it be supplied in each call to that routine. The "subscriber" and "utility-punch" parameters to the top level program are discovered by this means.

2. *Dynamic reference.* Similarly, the inference rule (sentence 6) contains a reference to "subscriber" that cannot be resolved within the context of the inference rule itself, but must be resolved instead in the larger context in which the rule is invoked. The inference rule is fired during the compare operation inside of "screen" (which is called by the top level program) when (and if) a match occurs. Hence, its dynamic context is the active portions of those routines. Within this dynamic context, resolving the "subscriber" reference is quite straightforward. It is simply a reference to the "subscriber" parameter of the top level program.

3. *Combination of separate conditional clauses.* Often, in informal descriptions a succession of IF-THEN statements (with no ELSE clause) specify the mutually exclusive actions to be performed for the various cases. Recognizing such mutual exclusion is, of course, critical to a semantic understanding of the specification and is dependent upon determining that the truthfulness of one of the predicates is sufficient to ensure that none of the others is simultaneously true. Because the prototype system has such a mechanism, it is able to combine the actions to be performed when the message is and is not addressed to the subscriber into a single IF-THEN-ELSE statement.

4. *Special action semantics.* We have attempted to remove all representation issues from the specification through our use of the relational data base. This has been highly successful, but some representation issues still remain. The notion of marking or noting something is such a case. Some value (the mark or notation) is associated with an object so that at some later point the existence of the association and particular value will

trigger some action. Naturally, the correspondence between a particular value and an action or goal is a representation issue. To remove this problem, the prototype system maintains a set of goals (or purposes) to be achieved at some later time and a set of marking values associated with these goals. When the informal specification specifies a marking operation without indicating the marking value, then the purpose of the marking operation is used by the prototype system to determine the appropriate marking-value. This construct occurs in the three marking operations in this example.

APPENDIX II

The prototype system is structurally quite simple. It has three phases (Linguistic, Planning, and Meta-Evaluation) that are sequentially invoked to process the informal specification. Each phase uses the results of the previous phases, but no capability currently exists to reinvoke an earlier phase if a difficulty is encountered. Hence, either ambiguity must be resolved within a phase or the possibilities passed forward to the next phase for resolution.

We will describe the prototype system by working backward from the goal through the phases (in reverse order) toward the input to expose the system design and provide context for understanding the operation of each phase.

The goal of the system is to create a formal operational specification from the informal input, which means that it must complete each of the partial descriptions in the input to produce the output. In general, each partial description has several different possible completions, and a separate decision must be made for each partial description to select the proper completion for it.

Based on the partial description and the context in which it occurs, an a priori ordered set of possible completions is created for each partial description. But one decision cannot be made in isolation from the others; decisions must be consistent with one another and the resulting output specification must make sense as a whole. Since the output is a program in the formal specification language, it must meet all the criteria for program well-formedness. Fortunately, programs are highly constrained objects (one reason they are so hard to write), so there are many well-formedness criteria which must be satisfied.

This provides a classical backtracking situation, since there are many interrelated individual decisions that in combination can be either accepted or rejected by some criteria (the well-formedness rules). In such situations, the decisions are made one at a time in some order. After each decision the object (program) formed by the current set of decisions is tested to see if it meets the

criteria (well-formedness rules). If it does, then the next decision is made, and so on, until all the decisions have been made and the result accepted. If at any stage the partially formed result is rejected, then the next possibility at the most recent decision point is chosen instead and a new result formed and tested as before. If all possibilities have been tried and rejected for the most recent decision point, then the state of the decision-making process is backed up to that existing at the previous decision point and a new possibility chosen. This process will terminate either by finding an acceptable solution (formal specification) or by determining that none can be found. The resulting object (program) is an acceptable solution (formal specification) for the problem (informal specification).

The order in which decisions (rather than the order of alternatives within a decision) are made should be chosen to maximize early rejection of infeasible combinations of decisions. This requires that the rejection criteria can be applied to partially determined objects. The preferred decision order is clearly dependent on the nature of the acceptance/rejection criteria.

We now let the nature of the well-formedness criteria determine the structure of the prototype system so that the early rejection possibilities inherent in the criteria can be utilized. The criteria fall into three categories: dynamic state-of-computation criteria, global reference criteria, and static flow criteria. Each of these categories must be handled differently.

The dynamic state-of-computation criteria are based only on the current "state" of the program and its data base (e.g., "the constraints of a domain must not be violated" and "it must be possible to execute both branches of a condition"). They require that all decisions that affect the computation to that point (but not beyond) must be made before the criteria can be tested. Thus, if decisions could be made as they are needed by the computation of the program and the program "state" examined at each stage of the computation, then the dynamic state-of-computation criteria could be used to obtain early rejection of infeasible decisions.

This is exactly the strategy adopted in the design of the prototype system. However, since no actual input data is available for the program to be tested, and since the program must be well-formed for a variety of inputs, symbolic inputs rather than actual inputs are used. Instead of actual execution, the program is symbolically executed on the inputs, which provides a much stronger test of well-formedness than would execution on any particular set of inputs.

However, completely representing the state of the computation as a program is symbolically executed is very difficult (e.g., determining the state after execution of a loop or a conditional statement) and more detailed than necessary for the well-formedness rules. Therefore, the prototype system uses a weaker form of

interpretation, called Meta-Evaluation, which only partially determines the program's state as computation proceeds (e.g., loops are executed only once for some "generic" element, and the effects of THEN and ELSE clauses are marked as POSSIBLE, but are not conditioned by the predicate of the IF). This Meta-Evaluation process is much easier to implement and still provides a wealth of run-time context used by the acceptance/rejection criteria to determine program well-formedness.

The global referencing criteria (such as "parameters must be used in the body of a procedure") test the overall use of names within the program and thus cannot be tested until all decisions have been made. They are tested only after the Meta-Evaluation is complete.

The final category of criteria, static flow (e.g., "items must be produced before being consumed" and "outputs must be produced somewhere"), are more complex. The Meta-Evaluation process requires a program on which to operate, which may contain partial descriptions that the Meta-Evaluation process will attempt to complete by backtracking. This program "outline" is created from the informal input for the Meta-Evaluation process by the flow analysis, or Planning, phase, which examines the individual process descriptions and the elaborations, refinements, and modifications of them in the input, then determines which pieces belong together and how the refinements, elaborations, and modifications interact. It performs a producer/consumer analysis of these operations to determine their relative sequencing and where in the sequence any unused and unsequenced operations should occur. This analysis enables the Planning phase to determine the overall operation sequencing for the program outline from the partial sequencing information contained in the input. It uses the data flow well-formedness criteria and the heuristic that each described operation must be invoked somewhere in the resulting program (otherwise, why did the user bother to describe it?) to complete the partial sequence descriptions.

If the criteria are not sufficiently strong to produce a unique program outline, the ambiguity must be resolved either by interacting with the user or by including the alternatives in the program outline for the Meta-Evaluation phase to resolve as part of its decision making process. In the prototype system, the Meta-Evaluation phase is prepared to deal with only minor sequencing alternatives such as the scope of conditional statements (if a statement following a conditional assumes a particular value of the predicate, it must be made part of one of the branches of the conditional) and demons. (Are all situations which match the firing pattern of a demon intended to invoke it or only those which arise in some particular context, and if so what context?) Major sequencing issues--such as whether one statement is a refinement of another or not--that cannot be resolved by the Planning phase must be resolved by the user before the Meta-Evaluation phase.

Both the Planning and Meta-Evaluation phases use a structural description of the application domain to provide context for their program execution, and inference rules that define relation interdependencies in the process domain. This structural base is the application-specific foundation upon which the Planning and Meta-Evaluation phases rest, and must be provided before they are invoked. It contains all the application-specific contextual knowledge. It augments the system's built-in knowledge of data flow and program well-formedness and enables the system to be specialized to a particular application and to use this expertise in conjunction with its built-in program formation knowledge to formalize the input specification.

The construction of a suitable application-specific structural base is itself an arduous, error-prone task. Furthermore, our study of actual program specifications indicated that most of the structural information was already informally contained in the program specification. We therefore decided to allow partial descriptions in the specification of the structural base and to permit such descriptions to be intermixed with the program specification.

Since we are concerned only with the semantic issues raised by using partial descriptions in the program specification, the system uses a parenthesized version of the natural language specification as its actual input to avoid any syntactic parsing issues. This parenthesized input does not affect the semantic issues we have discussed.

The first tasks, then, of the system are to separate the process descriptions from the structural descriptions, to convert both to internal form, and to complete any partial structural descriptions. These tasks comprise the system's Linguistic phase, which precedes the other two.

If a formal structural base already exists for some application, then, of course, it is loaded first and is augmented by and checked for consistency with any structural statements contained within the program specification.

Thus, in chronological order (rather than the reverse dependence order used above), the system's basic mode of operation consists of reading an input specification, separating it into structural and processing descriptions; completing the structural descriptions and integrating the result into any existing structural base; determining the gross program structure by producer/consumer analysis during the Planning phase; and, finally, determining the final program structure through Meta-Evaluation.

2. MILITARY MESSAGE EXPERIMENT

Research Staff:
Robert H. Stotz
David Wilczynski
Don Oestreicher
Wrenwick Lee
Paul Raveling
Leroy Richardson
Jeff Rothenberg
Ron Tugender
Elaine Sonderegger

Support Staff:
Joan Malone
Jeanne Ramirez

Contributing Staff:
Vernon Dieter
George Dietrich
Nelson Lucas
Robert Parker
Chloe Holg

INTRODUCTION

The Military Message Experiment (MME) is a joint program funded by DARPA and the Navy, taking place at CINCPAC (Commander-in-Chief, Pacific Headquarters). An operational test of an experimental, on-line interactive message handling system will be conducted at Camp Smith, Oahu, Hawaii, by a selected set of users from CINCPAC's Operations Directorate (J3), including the Command Center and the groups supporting it. The test calls for 25 CRT terminals plus seven electrostatic printers, distributed appropriately in the Command Center and the offices of the J3 staff, connected to a PDP-10 running the TENEX operating system and the SIGMA message service developed by ISI. The experiment is planned to last until October 1, 1979.

The test is intended to evaluate the utility of an interactive message service and study its impact on the user organization; in addition, it will provide much insight into how future message systems for this type of environment should be designed and built. Experience with civilian users of interactive message systems (e.g., the ARPANET community) has shown that the basic fabric of communication within the user organization has often been altered, though the nature of the change depends on such variables as who else is on the service, accessibility of the terminals, difficulty of learning the system, reliability of the service, features offered, response time, etc. A primary goal of MME is to learn the effect of these variables on the message service in the command and control environment. With this information, the various military organizations responsible for communications will be more effective in specifying the requirements for future production systems.

ISI became involved in this experiment in 1973 when it performed a study of the communication needs on the island of Oahu [Ref 1]. In late 1973 the Information Automation (IA) Project was formed to investigate the application of automation to military communications. The concept of conducting an experiment in an operational military command evolved slowly. In December 1975 a Memorandum of Agreement was signed by DARPA, COMNAVTELCOM, COMNAVELEX and CINCPAC, defining the MME. At that time three DARPA contractors, including ISI, were working on candidate message systems for MME. In February 1977 an extensive evaluation of the three systems was made and ISI's SIGMA message system was selected. The PDP-10 computer and five user terminals were installed in May 1977 at CINCPAC, and SIGMA was brought up.

During the next 12 months the system was shaken down and gradually improved. A few "friendly" users were introduced to the system and their evaluation influenced the direction of subsequent SIGMA development. Experience was gained in how to train users, resulting in increased emphasis in on-line self-contained training aids. By April 1978 SIGMA was receiving all of the J3 AUTODIN traffic (approximately 700 messages daily) on a 24 hour a day basis, 10 terminals were in user's offices, and use of the system by operational personnel was beginning. Regular Action and Information assignments on the incoming traffic began in July, and since that time use of the system has increased significantly.

In parallel with the development of the message service, the Navy and ARPA through MITRE Corporation have been defining the test objectives for MME and the plan for achieving these [Refs. 2 and 3]. This plan has required that SIGMA contain a Data Collection Facility (DCF) to log pertinent data about the use of SIGMA. This data is recorded in TENEX files on a continuous basis, and at regular intervals it is dumped from files to magnetic tape and sent to MITRE for analysis.

PROGRESS DURING THE REPORTING PERIOD

The fundamental design of SIGMA and the terminals was established several years ago. An overview of system design and implementation, as well as terminal development, through 1977 are given in [4 and 5]. Preliminary thinking about the design of this system can be found in Refs. 6-14. The principal task since then has been SIGMA's implementation as an efficient, responsive system capable of being run 24 hours a day, 365 days a year. In March 1977, when SIGMA was chosen over two competitive systems to be the message service for MME, at the time it was deficient in several aspects. A number of desired functions had not been implemented, and the system's response was far too slow to support 25 users. After SIGMA was brought up at CINCPAC in May 1977 and users began using the system,

another deficiency became apparent. The Daemons, as they were implemented, were too fragile to be be maintained on a round-the-clock basis. This occurred because they were designed very early on before their requirements and functions were well understood.

The bulk of the effort expended during the reporting period was directed toward bringing SIGMA to an operationally acceptable level. This was done while continuously running the CINCPAC system for user introduction and training. From July 1, 1977, through June 30, 1978, ten releases of new software were installed.

The first three months of the period from July to September 1977 were focused primarily on functional deficiencies. In the fall and early winter, the main emphasis was shifted to improving performance. The last four months were chiefly devoted to maintainability. We have been successful in all of these endeavors.

Functions

A number of functional improvements were made during the reporting period to bring SIGMA to the point where it is considered operationally usable. As the system was used, the importance placed on the various functions changed, so that some features considered desirable at the time of SIGMA's selection are still not implemented, while others, not considered then, have been installed. The following features were added during the past year: data collection, lessons and exercises, security and accountability features, function keys, archive and the ROUTE instruction.

The data collection facility automatically gathers the information for analysis of the use of the message service at CINCPAC. Data collection is performed in the User Job. Each instruction and function key executed by the user is logged into a special data collection file in each user's directory along with appropriate data for analysis. Additional data, called Points, are recorded at special points throughout the execution of certain commands. A simple data reduction program is provided for quick looks at these data collection files.

One of the challenges of MME is to train users without taking them away from their work for classroom training. The approach taken is to make SIGMA as self-instructive as possible, and let the users train themselves on-line at their convenience. SIGMA provides interactive training facilities in the form of on-line lessons and exercises. Whenever a user wishes, he may issue the LESSON instruction with the number of the lesson he wants to take. This enters him into a tutorial session describing a particular facet of the service. At appropriate points in

the lesson the user is invited to try an exercise. He does this by executing the EXERCISE instruction with the number of the exercise to be conducted. The exercise has a tutorial phase which first explains the instruction to be executed, then suggests the user try it. When the user executes the instruction, SIGMA performs the command on a dummy data base, thereby protecting the user's real data. The user may switch between the tutorial text and the command results with simple key pushes. The entire service is covered in a series of 12 lessons and 23 exercises.

The original design of SIGMA used the function keys of the terminal simply as alternates to typing. Each key would print in the Instruction Window an assigned text string of either a SIGMA instruction or a SIGMA parameter word. Instruction keys were grouped together, parameter keys were grouped in another set, and security-related keys formed a third group. To execute the DISPLAY NEXT ENTRY instruction the user pressed the two function keys DISPLAY, NEXT ENTRY, then pressed EXECUTE. Upon EXECUTE, SIGMA would parse the command as though the user had typed DISPLAY NEXT ENTRY in the Instruction Window.

Although this seemed like a simple, consistent command interface that mimimized typing requirements, slow system response made it unwieldy. After each key push, the user had to wait for SIGMA to interpret the key, write the text string into the Instruction Window, and return the cursor to the terminal before the next key could be pushed. It was faster to hunt and peck to type the instruction into the Instruction Window, since it required computer interaction only on the EXECUTE key. In August 1977, the Command Language Processor in SIGMA was changed to interpret the function keys as full instructions to be executed. Now a single function key initiates an instruction like DISPLAY NEXT ENTRY. As soon as the key is pushed, SIGMA can proceed to execute without the normal parsing and with none of the delay of writing to the Instruction Window, returning the cursor and waiting for the next key push. This change improved SIGMA performance in terms of CPU seconds required to execute a command and user performance in real time required to perform a given task.

In the CINCPAC environment, an apparently insignificant message which could have been received months earlier may suddenly become critical information. Since the system has only enough secondary memory to store about 20 days' worth of traffic, every message received must be archived in off-line storage for potential later retrieval. In June 1978 an archive facility, based on the TENEX BSYS system used to archive TENEX files, was added to SIGMA. It writes onto archive tape and deletes from disk any message that has not been referenced in n days, where n is a variable specified by the operator.

Archiving a message does not affect file entries that may point to that message. If a user asks to display the message (normally through some file entry), he is told it is archived and to push YES or NO function keys to indicate whether it should be retrieved. Retrieval of a message produces a RETRIEVAL citation to the message in the user's Pending File.

A key user of SIGMA is the administrative aide (titled J301) who assigns messages for action, forwards them for information, and builds readboards for nearly all incoming AUTODIN messages to SIGMA. Since he deals in such a large number of messages (500-700 per day), J301 needs all the help he can get.

The ROUTE command was invented to combine all the operations performed on a message by J301 (Action, Forward, File, Delete) into a simple one-step process. ROUTE takes as an argument a text object which contains specially formatted specifications for the Action, Forward, File and Delete operations to be performed. Most of these Route Lists are permanently stored in the system, but they can be built dynamically as they are needed. ROUTE accepts lists of file entries, which speeds J301's job tremendously, since SIGMA's retrieval instructions (RESTRICT, AUGMENT) make it easy to collect messages into classes that get the same assignments. Early experience indicates he can process his traffic four times faster with SIGMA than he can manually.

Performance

Many changes were made to improve system throughput. Initially intuition was the only guide to increasing the system's speed. Later Bolt Beranek and Newman supplied measurement tools that allowed us to quantitatively evaluate performance improvements in terms of CPU seconds saved for a given instruction. Other tools allow us to measure system overhead and identify the limiting computer resources (e.g., central processor cycles, page swapping, disk channel bandwidth, file access to new data). These tools have helped identify where to focus our efforts.

The results of these measurements showed that SIGMA limits first on CPU cycles, but it also has a very high page swapping rate. This indicated the program was too big for the memory allotted (512K words) and did too much computation, even for simple operations. Over three-quarters of the CPU cycles were spent in TENEX, either in the monitor or executing JSYS calls. A later study and analysis done with the cooperation of BBN identified the actual time required by each JSYS. This was the first time this information had been available. The system was changed to take advantage of what was learned from the JSYS study, which improved performance by about 25 per cent.

The most significant change to SIGMA for performance purposes was the introduction of a global text package, which allows the multiple forks of a job to refer to text through nonvariant pointers rather than having to copy the text from fork to fork. This change implied a new representation of messages and user files on disk and completely different processes for handling text.

Coding efficiency changes generally involved simplifying the mechanisms to accomplish specific tasks. For example, when SIGMA originally opened a file for a user, it created in primary memory the entire Virtual Terminal (VT) image of the file. Now SIGMA generates only as much of the VT image as it displays. If the user scrolls to see a part of the file that is not in the terminal, SIGMA generates the VT image on demand.

Another technique implemented to give the user better response to his commands has been to move functions into background processing (i.e., to the Daemons). For example, when the user forwards a message for Action (execute an ACTION command), the system adds the new addressee to the Action field of the message, sends a citation to his Pending file, and sends a citation to a special file called the Action Log. All of this processing, which used to be done in the user's job, is now done by the Message Daemon. Note that the total amount of processing done by the system is the same, but that the user is freed to handle the next instruction much sooner. By having the Daemons in a separate process group (pie slice) from the user jobs, we can tune the average queue length for the Daemons.

Maintainability

As SIGMA began to be used on a round-the-clock basis in late 1977, it became evident that the background Daemons were not adequately designed for maintainability. Because these processes do not have a user watching them, they must be very robust, capable of generating complete yet appropriate logs to allow thorough post mortems of any problems. Since the Daemons were designed early in SIGMA's history, when the requirements were not well understood, many mechanisms provided turned out to be unnecessary and made the code unduly complex. Release 2.0 in June 1978 primarily dealt with a complete reorganization of the Daemons. As a side benefit they were made smaller and considerably faster.

The new Daemons, implemented for Release 2.0, provide the same basic functions as their predecessors (except for the Archive Daemon, which provides a new function). However, each Daemon has a much simpler process structure, consisting of a top Program Control (PC) fork and one working fork that performs the processing of the Daemon functions. Those Daemons that must alter messages and/or user files may also contain a lower fork of a File Access Module or Message Access Module, which accesses files or messages, respectively. Not only is this

structure simpler than before, but much of the code is now common across all Daemons and is shared; this simplifies maintenance and reduces the number of pages to be loaded. A simple queue replaces the old Daemon interface, which relied on synchronous communication by means of TENEX signals. Now User Jobs are completely independent of Daemons and can run whether the Daemons are up or down.

The new Daemons use a new, more flexible error logging and tracing package. They now produce separate logs for the different individuals interested in their performance. All successful operations are reported in a Log file. Errors are reported in both a SHORT-ERROR LOG, where an operator can get a quick summary of problems, and a LONG-ERROR LOG, where programmers can look at the detailed state of the process at the time the error occurred. A Trace file contains even more abundant information for the programmer. In addition, any unusual condition detected by a program may be reported to the operator directly.

FUTURE WORK

In July 1978 limited experimental use began. In this phase of the experiment users are free to use the message service for whatever they wish. J301, who does Action assignment and distribution of J3's traffic, processes all traffic received so that Action officers have electronic equivalents of their readboards, and inital Action assignments are recorded by SIGMA.

New software releases in September and December of 1978 are intended to provide further performance improvement and several new functions. A new mechanism will be provided for alerting users and giving them immediate access to new messages as they arrive. Discretionary access controls will be provided for users' files and for messages. As the users gain experience with the service, they are suggesting other new features and facilities to allow them to be more productive with SIGMA. Already the list of desired enhancements includes such things as a facility for users to highlight text, a restructuring of memo formats, and changes to our current model for coordination. Each of these improvements will be considered; those that seem reasonable will be made. It is precisely this sort of feedback and our ability to respond to it that distinguishes this experiment from the more usual production system installation.

MME has been operating for the past year and a half with contractual responsibility for the system split between ISI, who supplies the application software (SIGMA) and the terminals, and another contractor, who provides and maintains the computer hardware with the TENEX operating system and the PDP-11 terminal controller. This divided responsibility has not been effective in

tracking down problems that arise or in coming up with creative ideas for improving overall system peformance. In October 1978, ISI took over full system responsibility. In addition, a new KL computer system was installed to replace the two KA's, which has doubled throughput at least. The KL runs KI Emulation microcode.

REFERENCES

1. Ellis, T. O., J. F. Heafner, L. Gallenson, and J. T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, USC/Information Sciences Institute, ISI/RR-73-12, May 1973.

2. Goodwin, N. C., J. Mitchell, and S. W. Slesinger, *Test Plan for Military Message Handling Experiment*, Volumes I and II, MITRE Corporation, MTR-3268, July 1976.

3. Goodwin, N. C., and S. W. Slesinger, *Test Procedures for Military Message Handling Experiment*, MITRE Corporation, MTR-3521, October 28, 1977.

4. *1977 Annual Technical Report: A Research Program in Computer Technology*, USC/Information Sciences Institute, ISI/SR-77-8, September 1977.

5. Stotz, R., R. Tugender, D. Wilczynski, and D. Oestreicher, "SIGMA: An Interactive Message Service for the Military Message Experiment," *1979 AFIPS National Computer Conference Proceedings*, (to be published).

6. Tugender, R., and D. R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, USC/Information Sciences Institute, ISI/RR-74-23, May 1975.

7. Rothenberg, J. G., *An Intelligent Tutor: On-line Documentation and Help for a Military Message Service*, USC/Information Sciences Institute, ISI/RR-74-26, May 1975.

8. Heafner, J. F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, USC/Information Sciences Institute, ISI/RR-74-21, September 1974.

9. Abbott, R. J., *A Command Language Processor for Flexible Interface Design*, USC/Information Sciences Institute, ISI/RR-74-24, September 1974.

10. Rothenberg, J. G., *An Editor to Support Military Message Processing Personnel*, USC/Information Sciences Institute, ISI/RR-74-27, June 1975.

11. Heafner, J. F., *Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings*, USC/Information Sciences Institute, ISI/RR-75-34, July 1975.

12. Heafner, J. F., M. D. Yonke, and J. G. Rothenberg, *Design Considerations for a Computerized Message Service Based on Washington, D.C., Navy Personnel*, USC/Information Sciences Institute, ISI/WP-1, May 1976.

13. Heafner, J. F., and L. H. Miller, *Design Considerations for a Computerized Message Service Based on Triservice Operations Personnel at CINCPAC Headquarters, Camp Smith, Oahu*, USC/Information Sciences Institute, ISI/WP-3, September 1976.

14. Heafner, J. F., L. H. Miller, and B. A. Zogby, *SIGMA Message Service Reference Manual*, USC/Information Sciences Institute, ISI/WP-5, February 1977.

3. PROGRAM VERIFICATION

Research Staff:
Ralph L. London
Raymond L. Bates
Susan L. Gerhart
David R. Musser
David H. Thompson
David S. Wile

Research Assistants:
Roddy W. Erickson
David G. Taylor

Support Staff:
Lisa Moses
Betty Randall

INTRODUCTION

Our work concerns the process of constructing computer programs in ways that make it possible to verify that each program meets its specifications. Given some description of a programming task, it is ultimately necessary to construct two things: a formal specification of the task and a program for carrying out the task. The central ingredient of our work is the ability to verify the program, i.e., demonstrate by a mathematical proof that the specifications and the program are consistent with each other. The style of specifications and the methods of program construction, especially from isolatable program components, are all important influences in making verification feasible. Our work emphasizes the construction of tools to aid verification, specifically (1) tools to produce theorems sufficient to express consistency while retaining the relationship between the theorems and the individual parts of the program, (2) tools to prove the required theorems with the tools being both sufficiently powerful and sufficiently natural to be guided by humans, and (3) tools to accommodate changes in programs, specifications, assumptions, and proof steps that are essential to successful program construction and verification.

One example from among numerous instances where it is vitally important that computer software perform according to its specifications is the Security Kernel being developed for the Unix operating system by G. J. Popek and his colleagues at UCLA. It is very important that the Kernel permits access to privileged data only when an individual is allowed an access, i.e., that it maintains the specified form of data security. A verification using a system developed by this project is being constructed to demonstrate that the Kernel meets such a specification.

To support such applications of program verification to critical software components, the goals of our work are to

- develop a comprehensive, well-integrated set of techniques for constructing fully specified and verified software components, from which large, reliable, and modifiable software can be economically produced;

- implement the techniques in a widely accessible, user-oriented, interactive computer system; and

- demonstrate the usefulness and practicality of the techniques and system in applications to real software where reliability is of critical concern.

ACCOMPLISHMENTS

Techniques and system implementation

Our research at ISI has contributed significantly to the understanding of many of the fundamental issues underlying program specification and verification and to the development of effective techniques for dealing with these issues. These techniques include (1) a methodology for employing abstraction and hierarchical program organization and (2) extended system and language facilities, including improved specification and theorem proving techniques. We have implemented many of these new techniques in an experimental verification system called Affirm. We first discuss the methodology and then focus on some particular features of the Affirm System.

1. Abstraction and hierarchical program organization. A key to the ability to verify reasonably large programs is the use of abstraction. Programs can be hierarchically organized into smaller, segregated components which can be specified by means of abstractions - retaining essential properties at the level at which they are important while ignoring inessential details. A major concern of our research has been the development of verification methods that can take advantage of hierarchical structure in programs and specifications. At the proper level the necessary details are verified once per definition of the abstraction rather than once per use of the abstraction.

Structuring programs by using abstractions also has important consequences in the area of program modification. If changes in representations of abstractions can be made without changing the abstract properties (as is often the case), it is necessary to reverify only the definition and not each use of the abstraction.

Indeed, we have reverified modified programs by changing the proofs only in ways corresponding to the program modifications; it is unnecessary to reverify the entire program from the beginning.

The techniques of abstraction and hierarchical program organization are central to the design philosophy of the Alphard language [Alphard] (and to Clu also [Clu]). The techniques are therefore specific enough to be used in actual, albeit experimental, programs.

2. Some features of the Affirm System. The Affirm System accepts specifications of data abstractions and programs in a formal specification language that can be dynamically extended to new application domains. The system accepts programs written in a variant of the Pascal language (extended with abstractly defined data types and some other features of the Euclid language), together with embedded assertions. The Affirm verification condition generator supports the standard inductive assertions method and the more recently developed subgoal assertions method. Affirm also contains a natural deduction theorem prover for interactive proof of verification conditions and of properties of data abstractions. Lastly, Affirm is capable of organizing large specifications and collections of axiomatic and derived properties in an online data base for easy retrieval during subsequent program or data abstraction verifications. (Affirm is written in Interlisp and runs on a DEC PDP-10 computer at ISI. It is the successor to, and combines features of, two previous systems developed at ISI, the Xivus System and the Data Type Verification System.)

Affirm provides a specification language which has features in common with programming languages (such as its use of familiar conventions for expression syntax and declaration structure) and with logical languages (such as first order predicate calculus). The use of data abstraction is supported by the structure of the specification language and by the capabilities of the theorem prover component of the system. The system directly supports the algebraic axioms method of specification and also easily accommodates another method we have been studying, that of abstract model specifications (the use of a representation of the data in terms of some well-known mathematical objects such as sets, sequences, or tuples). The user can define these abstractions (or access them from a library) as well as other abstractions that are particularly suited to his application domain. In conjunction with the theorem prover commands, the user can build up a useful base of axioms and derived properties about data abstractions that can be invoked in later proofs. The involvement of those abstractions in proofs may come about either by their use in expressing programs--e.g., a program which manipulates a queue or file--or in expressing specifications of programs such as in the inductive assertions used to express relations between variables used by programs--e.g., the use of sequences to describe the list of values processed during the execution of a loop.

A concept which is extremely important to understanding the Affirm specification language is the role of equations in specifications and proofs. Affirm encourages the use of equational specifications of data abstractions, since the theorem prover is oriented toward performing deductions insofar as possible by making equational substitutions. The theorem prover is able to conduct such deductions automatically and efficiently by treating equations, whenever possible, as rewrite rules. These are rules of the form *left -> right* where *left* and *right* are expressions containing variables; these rules are used to rewrite expressions by replacing any subexpression that is matched by *left* with a corresponding version of *right* (i.e., with the same substitutions for variables that were made in matching *left*). Rewrite rules are applied to an expression until no further rewriting is possible. Thus, an essential property of a set of rewrite rules is *finite termination*, i.e., that no infinite sequence of rewrites is possible. Another extremely useful property is *unique termination*, i.e., that any two terminating sequences of rewrites starting from the same expression have identical final expressions (no matter what choice is made as to which subexpression to rewrite or which rule to apply first). A set of rules with the finite and unique termination properties is said to be *complete*. If a set of axiomatic equations can be treated as rewrite rules, and if these rules or a finite set of rules derived from them are complete, then one can decide when an equation is provable from the axioms just by rewriting both sides to their final expressions and checking for identity. Using rewrite rules to prove equational properties is generally much more efficient than other techniques which require heuristic searching. Thus the Affirm system attempts to form the parts of specifications which are equations into rewrite rules which have this completeness property. For this purpose a version of the Knuth-Bendix algorithm for completing a set of rewrite rules [Knuth-Bendix, Musser-Texas paper] has been implemented and made an integral part of the processing of specifications input to the system.

This completion algorithm has proved extremely useful in the construction of complete sets of rewrite rules for data abstractions such as sets and sequences. In these constructions, one starts with a set of equational axioms of a type along with one or more equations that are inductive theorems of the data type (i.e., provable from the axioms and an induction principle). The completion algorithm treats these equations as rewrite rules and determines additional rules that combine with the given ones to form a complete set. This complete set will automatically accomplish many proof steps that otherwise would require the user to carry out a proof by induction.

The theorem prover component of Affirm is a new prover which replaces the Bledsoe-Bruell prover used in the Xivus System. Its central design feature is the use of rewrite rules for many purposes, from performing basic propositional calculus proofs to searching for proofs by contradiction by using the Knuth-Bendix

completion process to check for consistency. It is a natural deduction theorem prover in the sense that it supports a natural style of search for proofs by setting up goals, splitting into subgoals, and displaying current subgoals in a form as close their original form as possible, subject to the simplifications that are performed. However, it does provide two methods of searching for proofs that are oft effective when subgoaling has produced current goals that are provable by instantiation of variables: the method of searching for proofs by contradiction mentioned above, and a method called chaining and narrowing [Lankford- Musser]. The latter method is applicable when there exists a single substitution instance of the current goal that is a tautology in propositional calculus.

The Affirm system has with human guidance in some cases successfully proved a series of examples including sets, symbol tables, sequences, marking algorithms, plus parts of the Security Kernel and (see below) the Delta File. In addition, the system has served as an evaluator of research ideas, and perhaps more importantly, as a generator of research questions such as proof strategies and methods for ensuring completeness and consistency.

Verification impact on Euclid, Alphard, and DoD language designs

In addition to applying verification ideas and methods to verifying programs and to constructing the verification systems, we have been concerned with applying the ideas, methods, and point of view to the design of new programming languages. We have participated in the Euclid language effort [Euclid] which produced a language for systems programs which are to be verified. An important part of the effort was the construction of Euclid proof rules [Proof Rules], an activity which led to an increased understanding of Euclid and to a number of improvements to the language. Except for machine dependencies, all constructs of Euclid are in principle verifiable with existing techniques.

In turn, a number of the ideas in Euclid have been incorporated into the Ironman specifications for DoD language designs [DoD]. We completed an evaluation of all four preliminary designs for the languages, mainly from the viewpoint of verification.

In the study of data and control abstractions, information hiding, and hierarchical program organization in the Alphard language project, verification concerns were truly symbiotic with programming methodology concerns. The Alphard language is heavily influenced by the goal to produce verified programs, for example the direct inclusion in the program of specifications (both abstract and concrete) for each component, the appearance of clauses to check certain parameters, and the simplified versions (compared to previously constructed ones) of control constructs such as generators. Besides the basic *form* mechanism and the

iteration and generator constructs, additional features of Alphard that are essentially completely designed include selectors (special cases are array subscript and record field selection), reference parameters of procedures, and parameter binding of all actual parameters. The designs include the required proof rules. Programs and proofs may now include the standard notion of "location" such as array element or node of a tree. Indeed the Alphard verification methodology has been one of the main proof techniques used in verifying the Security Kernel and the Delta File.

Real-world, relevant, verified examples

UCLA Security Kernel. As mentioned in the Introduction, the Security Kernel is being developed at UCLA for the Unix operating system. The Alphard verification methodology is being applied to the data structure abstractions of the Kernel. All of the abstract specifications and concrete specifications are complete. The concrete-to-abstract mapping is nearly complete; the proof of the relationship between the concrete and abstract invariants is about one-half done. The proof of the concrete entry assertions is (trivially) complete. The proofs of the abstract exit assertions and the proofs across the implementation code are complete for several of the operations of the Kernel.

The Delta File. The Delta File is a component of SIGMA, the operating system for ISI's Military Message Experiment. Briefly, the task is to permit several users to modify a single central file simultaneously, subject to certain constraints. The Delta File code has been thoroughly analysed to produce (1) very high level prose and Alphard-like specifications of the system and task, (2) algebraic axioms which describe the tree-like and sequence-like data structures employed in the implementation, (3) recursive functions that succinctly express the algorithms, (4) Pascal programs with the abstract data types which more concretely simulate the actual SIGMA code (written in Bliss), and (5) Alphard-like abstract-to-concrete links between the Pascal and Bliss programs. The overall specification uses a mixture of specification methods--algebraic axioms, Alphard, mathematical notation, and prose--that reveals some of the strengths and weaknesses of each method.

There are four verification tasks: (a) an informal argument that if certain properties hold about the functions of (3) then the requirements in (1) are satisfied, (b) an Affirm proof that the properties of (a) do hold, (c) Affirm proofs of the Pascal programs of (4) which show that the functions of (3) are computed by the programs (including verification condition generation and proof), and (d) informal proofs (by a Bliss expert) of the abstract to concrete links in (5). Again the verification is a mixture of informal and mechanical, dealing both with programs and with properties outside the programs which show that requirements are satisfied.

CONCLUSIONS

This experiment and the Security Kernel experiment have shown that formal specification and verification methods and tools do apply to real, intermediate-sized, moderately complex software components. We have learned a great deal about the strengths and weaknesses of these methods, the improvements that will be needed in our tools, and the resources required for such a specification and verification effort. Our conclusion is that formal specification and verification on a task like this is quite feasible, and that the main task now is to improve our tools and acquire the experience to make this type of programming activity more practical. An overview of problems and prospects is presented in [Gerhart].

REFERENCES

[DoD] DoD High Order Language Working Group, Department of Defense Requirements for High Order Computer Programming Languages -- Revised "Ironman," July 1977.

[Gerhart] Gerhart, S. L., *Program Verification in the 1980s: Problems, Perspectives, and Opportunities*, USC/Information Sciences Institute, ISI/RR-78-71, August 1978.

[Alphard] Hilfinger, P., Feldman, G., Fitzgerald, R., Kimura, I., London, R. L., Prasad, K., Prasad, V., Rosenberg, J., Shaw, M. and Wulf, W. A. (editor), *An Informal Definition of Alphard* (Preliminary), Carnegie-Mellon University Technical Report, February 1978.

[Knuth-Bendix] Knuth, D. E. and Bendix, P. G., "Simple Word Problems in Universal Algebras," *Computational Problems in Abstract Algebra*, Leech, J. (editor), Pergamon Press, New York, 263-297, 1970.

[Lankford-Musser] Lankford, D. S., and Musser, D. R., On Semideciding First Order Validity and Invalidity, submitted for publication.

[Euclid] Lampson, B. W., Horning, J. J., London, R. L., Mitchell, J. G., and Popek, G. J., "Report on the Programming Language Euclid," *SIGPLAN Notices*, 12, 2, February 1977. A revised report is in preparation.

[Clu] Liskov, B., Moss E., Schaffert, C., Scheifler, B. and Synder, A., *Clu Reference Manual*, MIT Laboratory for Computer Science Computation Structures Group Memo 161, July 1978.

[Proof Rules] London, R. L., Guttag, J. V., Horning, J. J., Lampson, B. W., Mitchell, J."G., and Popek, G. J., "Proof Rules for the Programming Language Euclid," *Acta Informatica*, 10, 1, 1-26, 1978.

[Musser-Texas paper] Musser, D. R., "A Data Type Verification System Based on Rewrite Rules," *Proceedings of Sixth Texas Conference on Computing Systems*, Austin, Texas, Section 1A, 22-31, November, 1977.

4. NETWORK SECURE COMMUNICATION

Research Staff:
E. Randolph Cole
Stephen Casner
William Fisher
James Koda
Eric Mader
Gertrud Mellstrom
Seth Michaelson

Consultant:
Danny Cohen

Support Staff:
Debe Hays
Robert Parker
George Dietrich
Oralio Garza
Clarence Perkins
Leo Yamanaka

INTRODUCTION

For the past several years, the ISI Network Secure Communications (NSC) Project, along with the other contractors in the ARPA NSC Group, has been implementing prototype packet voice transmission systems on the ARPANET and other packet networks. The great success of this work in the prototype stage has raised many questions for the future. Does it scale up? What will happen if a packet voice network is built with thousands of extensions? A study by the Network Analysis Corporation [1] concludes that packet voice will scale very well economically; indeed, the study predicts that an integrated packet data/voice network will be the lowest-cost means of serving long-range DoD communications needs.

But a question remains: Will packet voice systems scale up technologically? To provide the tools and experience to answer this question, ISI and the other ARPA NSC contractors are working on bringing packet voice out of the prototype stage, creating new software, hardware, and protocols for low-cost communications in the packet network environment of the future.

During the past year, the NSC effort at ISI has concentrated on making real-time measurements of packet voice, designing generalized structures for interfaces to packet voice systems, and laying the groundwork for participation in the satellite-based high bandwidth packet voice experiment.

APPROACH

A critical feature of any work in packet voice--whether protocols, software, or hardware--is flexibility. New hardware developments and accompanying increases in vocoder capabilities are happening so fast that any packet voice system or protocol that is inflexible or lacking in generality will be left behind. The same is true of the networking aspects of packet voice; new types of packet networks and new implementations of existing types are coming thick and fast. While complete and total flexibility and generality is not only impossible but surely extremely inefficient, care must be taken to provide for foreseeable changes, improvements, and expansion. For example, in the design of the Network Voice Protocol (NVP) and Network Voice Conferencing Protocol (NVCP) the ARPA NSC group took care to make the protocols as independent as possible of the specific vocoder used. Matters specific to the particular vocoder in use are confined to one phase of the protocol; a new vocoder can be accommodated just by changing a data table.

Flexibility is particularly important in the testing and comparative evaluation of new vocoders and other packet voice improvements. During the next few years, many new vocoders are likely to be developed and will need to be compared and evaluated; thus network interfaces, protocols, and user interfaces must be readily available.

Another very important principle of the ISI NSC group has always been to maximize the usefulness of its products to the ultimate user. The principle is the same whether the product is a consumer product or a packet voice system for military use; if the system is difficult to use, it will not be used, no matter how careful the underlying design and implementation.

Since the beginning of packet voice experiments, the ISI NSC project has been aware of the three-way tradeoff between quality, bandwidth, and delay (response time) in packet voice systems. Changing any one of these three factors affects the other two. An effort has constantly been made to understand this tradeoff and exploit it in order to provide the best possible packet voice communication systems. As a result, a packet voice measurement program was established and is continuing in an effort to better understand the dynamic behavior of packet networks as it affects voice traffic.

RESEARCH AND DEVELOPMENT GOALS

There are at least three major goals for packet voice research and development: first, to extend its usefulness to all types of packet nets and allow communications between points on different, interconnected networks; second, to develop software protocols and hardware structures for large-scale (1000-phone) packet voice systems; and third, to greatly reduce the size and cost of packet voice terminals.

Therefore a major objective of the ISI NSC project is to develop a second-generation network voice protocol, called NVP-II. The new protocol will incorporate experience gained with the original NVP, the newer NVCP, and the latter's extension to allow use of the Packet Speech Measurements Facility (PSMF). The new protocol will be designed from the outset to operate in an internet environment and to support the multiplexing required for efficiency in a large-scale packet voice system. NVP-II will also support packet voice measurements in a different, better way than the present protocol. The ISI NSC project has already defined the requirements that NVP-II must meet and is currently working with the other ARPA NSC contractors in developing the protocol itself.

The second major objective will be to participate in the satellite-based high bandwidth packet voice experiment, using a satellite node to be installed at ISI in FY79. It is anticipated that the high-bandwidth satellite network will be the first with the capacity to support large-scale packet voice systems with hundreds or thousands of voice terminals. This will provide a major opportunity to develop high-quality user-oriented conferencing systems. The high bandwidth will also support initial efforts to develop multimedia communications systems, with graphics, facsimile, text, data and perhaps even video in addition to voice.

Another objective of the ISI NSC project is to aid in the effort to transfer packet voice technology out of the laboratory into small, low-cost packet voice terminals. This task will require careful adaptation of present networking techniques and protocols if the result is to be usable with many different vocoders and networks, including an internet environment. ISI's role in this effort will be to provide protocol, networking, and user interfacing expertise.

PACKET VOICE INTERFACING

One issue that had previously received little attention as a research topic was the issue of how to interface vocoders to packet networks. Although several different packet voice systems had been implemented on different packet nets, each

system's interface to the packet net was built with a particular network interface hardware and packet net in mind.

With future requirements for very low cost packet network interfaces as well as interfaces capable of handling hundreds of voice connections, it became apparent that there was a need to analyze the hardware and software components that form a packet voice interface (PVI). The need was not to design the "best" overall PVI, one which might handle all possible situations, but to analyze the task into its component parts in order to better understand the problem.

Requirements

There are a number of requirements the PVI must meet if it is to be as general as possible. Not all implementations of the PVI will need to meet all these requirements, but the general structure of the PVI must account for all of them. These requirements are as follows:

1. *Vocoder-independence.* It is anticipated that vocoders ranging from 2.4 kbps LPC to 64 kbps PCM will be encountered.

2. *Network-independence.* It must not rely on the characteristics of any one network.

3. *Protocol-independence.*

4. *Full-duplex.* It must support full-duplex communications.

5. *Conferencing.* It must support multi-user non-shared-air conferencing.

6. *Encryption.* Encryption of the voice data must be possible within the structure of the PVI.

7. *Multiple vocoders.* The structure of the PVI must provide for as many (possibly different) vocoders as the hardware and network can handle. This does not mean translation between different types of vocoders, however.

8. *Remote operation.* Because the vocoder may not be located at a network node, provision must be made for the vocoder to be remote via telephone or other lines. Encryption must still be possible. The term PVI means everything between the vocoder and the packet net, including the parts of it which are remote.

9. *User interface.* Its interface to the user must be simple and easy to use.

10. *Economics.* The structure should concentrate expensive parts in as few places as possible and let parts which will be built in quantity be as simple (and therefore low cost) as possible.

Block Diagram

Figure 4.1 is a block diagram of the PVI. There are three main sections in the block diagram. Each section is made up of several building blocks, each of which will be discussed in some detail. The interfaces between them will also be discussed.

For outgoing data the jobs performed by the sections are:

Section A: Transformation of voice and control information into digital form.

Section B: Assembly of the voice and control data into pieces called "frames". Conversion of control data into control frames structured according to a voice protocol.

Section C: Addition of network-specific addresses and headers, possible multiplexing of several voice frames into larger packets, and delivery of the packets to one or more networks.

For incoming data each section performs the reverse task.

This division of the task into three major sections greatly enhances generality, because the interfaces between the sections can then be clearly defined. Thus, for example, the vocoder/control section can be replaced with a different set of hardware in order to use a different type of speech compression.

A short summary of the blocks in Fig. 4.1 and their tasks is as follows:

1. *Control:* Implements the user interface. Transforms the user's signals into digital control signals, and incoming signals into human engineered alerting signals.

2. *Analyzer:* Transforms the analog voice waveform into a digital data stream.

3. *Synthesizer:* Transforms a digital data stream back into an analog voice waveform.

4. *Encoder*: Quantizes the digital data stream from the analyzer for transmission using fewer bits.

5. *Decoder*: Expands the encoded digital data stream back into the form which the synthesizer uses.

6. *Voice Protocol Controller*: Converts digital control signals from the user into packet-like frames containing NVP-compatible control information and vice versa.

7. *Packetizer*: Organizes the digital data stream into packet-sized frames and adds the NVP header (time stamp, length and silence indication).

8. *Depacketizer*: Removes the NVP header from incoming frames and produces a digital data stream with the proper timing.

9. *Concentrator*: Dynamically concentrates data and control frames into packets sized and addressed for efficiency and conference control. Performs the reverse task for incoming packets.

10. *Conference Controller*: Handles conference control for the concentrator and the vocoders connected to it.

11. *Internet Process*: Handles internetting host-to-host protocol issues.

12. *Local Network Driver*: Adds or removes the local network's header. Handles any network-specific functions, if necessary.

The reader should note that Section A contains all the knowledge about the speech compression algorithm, Section B contains all the knowledge about the voice protocol, and Section C contains all the knowledge about the packet network(s) and their protocols. This separation is not always strictly true, such as when the voice data encoding and decoding are done in Section B, but the separation is otherwise quite complete.

It should also be pointed out that Section B inputs and outputs frames, that is, roughly packet-sized pieces of data and of control, but without headers for any specific network. These frames are identical to the packets now sent on the ARPANET, without the ARPANET leader. The only information now contained in the ARPANET leader which would have to be put in the frame itself is one bit telling whether the frame is data or control information. The ARPANET Link field now carries that information, since data and control are sent on different LINKs.

Stating it another way, one can say that the voice protocol itself carries no information about the network, since it is a higher-level protocol than the network communication protocols. The network-specific information, in the form of the network's host-host protocol, can be put on the frame before it is sent and removed afterward. The voice protocol remains the same for all the networks, since it carries information about the voice communications, not the network.

This organization then permits the frames to be concentrated for efficiency, to be grouped into larger packets or messages for transmission of several voice streams going to one place, to be sent over several different networks, etc. The device which does this is called a concentrator. For a satellite network, the concentrator might be a large, fast piece of hardware capable of handling many vocoders. Alternatively, a single microprocessor might do the job in the case where only one vocoder is interfaced to the PRNET.

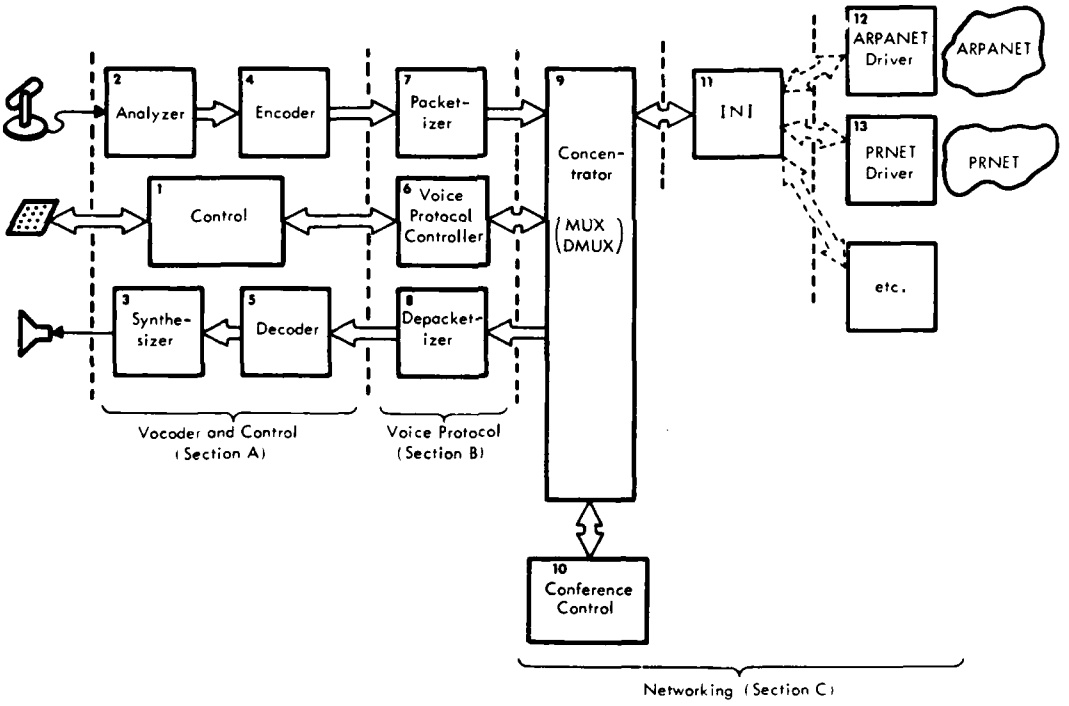


Figure 4.1 Packet voice interface block diagram

PACKET SPEECH MEASUREMENTS

The Packet Speech Measurements Facility (PSMF), located at the Computer Corporation of America (CCA) in Cambridge, Massachusetts, became operational in 1977. The purpose of the PSMF is to provide a measurement and storage facility for packet voice. The PSMF uses an extended version of the Network Voice Conferencing Protocol (NVCP), which allows a user elsewhere on the ARPANET to establish a connection, store and retrieve packet voice, and request the PSMF to perform certain measurements on the packet voice data, particularly measurements of the arrival time patterns of voice data packets.

During the last year the ISI NSC project has helped in the initial testing of the PSMF, suggested extensions to the PSMF to increase its usefulness, and used it extensively for packet voice storage and measurement. In addition to the simple storage of segments of packet voice, ISI has used the PSMF to store and retrieve "voice messages," using a simple prototype system for the task.

The voice message system uses both a packet voice terminal and an ordinary text-oriented terminal. The sender types in the destination and subject of the message just as he would with a normal text message, along with a password and file name used for access to the voice message. Then the sender dictates the body of the message using the packet voice equipment. Two items are then generated by the voice message system: a normal text message sent to the recipient's home directory, and a packet voice file at the PSMF. The text message contains the subject and the file name and the password the recipient needs to retrieve the packet voice message body.

It should be pointed out that this packet voice message system was designed primarily as a test and demonstration tool using the PSMF. A large number of issues remain in the area of voice messages, most important being whether or not such a system, which is a hybrid of text and voice, is really much more useful than the usual text message. Ideally, a voice message system would use speech recognition and voice response to make a system that needed no text input for destination, etc. The ISI NSC project will be investigating these issues in the future.

In the area of packet voice measurements, three major experiments were conducted. The first sought to determine the effect of network load on packet voice traffic. The second, the most interesting of the three, was designed to determine how to minimize delay from speaker to listener. The third was to check out an IMP software parameter that was thought to be a possible bottleneck for packet voice in the ARPANET.

The Effect of ARPANET Traffic on Packet Voice

The first experiment, and by far the most time-consuming, was designed to investigate the effect of other ARPANET traffic on packet voice transmissions. Three five-minute voice transmissions were made from ISI to CCA every weekday for about six weeks, at 2 PM, 4 PM, and 7 PM PST. These times were chosen as very likely to exhibit high, moderate, and low cross-country ARPANET traffic, respectively, under the seemingly universal assumption that there is a peak in network and processor usage at the end of the working day. Thus 2 PM PST represents roughly the end of the East Coast work day, 4 PM roughly the end of the West Coast work day, and 7 PM an evening load on both coasts. Whether or not these assumptions were exactly correct, it was anticipated that some trends would appear in the measurements.

The parameters measured were the numbers of missing, out-of-order, and duplicate packets, and the variance in the relative transit time across the ARPANET. It was (and still is) only possible to measure relative transit time, because both ends lack a highly accurate measurement of absolute time.

The results were somewhat surprising. There appeared to be no significant statistical difference in any of the measured values either with time of the day or day of the week. This was surprising because packet voice researchers still believe that other traffic does affect packet voice.

This lack of variation observed in the first experiment was explained in part by the second experiment. In the second experiment the rate at which voice data packets was given to the network was varied, and large changes in the delay and its variance during packet voice data transmissions were observed, especially when high packet rates were used. The packet voice data generated in this first experiment was given to the network in large packets at a low rate.*

Future experiments in the effect of network loading on packet voice will use a higher packet rate, at which the network is more variable. It will also be worthwhile to test packet voice segments made up of several shorter bursts of voice to see if the five-minute duration of these tests caused variations to average out.

*During the second experiment it was determined that the network handles large packets given to it at a low rate very well.

Minimizing Delay on the ARPANET

The second packet voice measurement was an effort to see how end-to-end delay of packet voice transmissions from the speaker to the listener can be minimized. The idea for the experiment was generated by observing packet voice conversations between ISI and MIT's Lincoln Laboratory in Massachusetts. Voice transmissions from ISI to Lincoln sounded relatively good at Lincoln, but transmissions in the opposite direction were very garbled because of out-of-order and lost packets. Although a packet network can exhibit such nonsymmetric behavior, the effect seemed too great to fit that explanation. It was noted, however, that Lincoln generally sent small voice data packets in order to minimize the time spent assembling the packets, while ISI used somewhat larger packets sent at a lower rate. A change in the size of voice data packets sent by Lincoln greatly improved the voice quality heard at ISI. Therefore an experiment was conducted to determine the effect of packet size on packet voice system performance and determine if there was an optimum size that minimized overall delay while maintaining good quality. During the experiment it was found that the primary factor was not the packet size, but the packet rate (inversely proportional to size in a fixed-rate system).

Overall delay between speaker and listener includes several factors. There is a relatively small delay in the LPC algorithm itself. The two major components are packet assembly time and network transit time. Packet assembly time is the period between the times the first and last parcels are placed in a packet. Network transit time varies between some minimum and maximum values, depending on network conditions. However, a reconstitution delay must be artificially inserted by the receiver to accommodate the variance in network transit time so that the reconstructed speech is continuous. Consequently, the effective network transit time for all packets is the same as that of the slowest packet if all packets are to be received in time for continuous reproduction.

In order to reduce packet assembly time, it is desirable to make packets as short as possible. However, since the data rate is constant, sending shorter packets implies that the packet rate will be higher. A higher packet rate causes more load on the network, increasing the maximum network transit time; to compensate, the reconstitution delay must be increased. Since the overall delay is the sum of these two opposing factors proportional to the packet rate, there should be a packet rate that minimizes the overall speaker-to-listener delay.

Experiments. Therefore the second major experiment was conducted by ISI with the cooperation of Lincoln Laboratory to determine the optimal packet rate for transmissions between the two sites. The two sites are separated by a minimum of

eight nodes. Speech packets were transmitted from Lincoln to ISI at various rates. Each packet contained the time of its transmission, which was recorded at ISI along with the time of receipt. Since no facility currently exists to accurately determine the absolute time difference between the clocks at the transmitter and receiver, only relative transit times could be calculated. (Relative times were sufficient, however, for these experiments.)

The data rate was 5000 bits per second throughout the experiment. The following packet rates and corresponding packet sizes were used:

Packet Rate (packets/second)	Packet Size (50-bit parcels/packet)
20.0	5
14.3	7
10.0	10
6.7	15
5.6	18

Data was recorded for several minutes of speech transmission at each of the packet rates on three different days. For each of the transmissions two graphs were plotted: The first showed the network transit time for each packet relative to the minimum transit time versus the time the packet was transmitted. The second was a histogram of these relative transit times.

Results. Figure 4.2 shows a 30-second segment of a high packet rate (20 packets per second) transmission. Each line represents a single packet, showing the amount by which its network transit time exceeded the minimum. The spaces between groups of packets correspond to silence intervals during which no packets were sent. Note that the variance in network transit time is nearly three seconds.

The improvement in network performance at low packet rates (5.6 packets per second) can be seen by comparing Fig. 4.3 with Fig. 4.2. The packet lines are spaced farther apart because the interval between packet transmissions is larger.

Figures 4.4 and 4.5 are histograms of the relative transit times for the complete transmissions from which the segments in Figs. 4.2 and 4.3 were extracted. Referring to Fig. 4.4, the minimum effective network transit time to receive 100% of the packets in time would be 2.98 seconds plus the actual transit time of the fastest packet. Similarly, 90% and 80% lines are drawn on the histogram showing the delay required to receive 90% and 80% of the packets in time. These three values were calculated from each histogram, and the results

were averaged over the three days' experiments at each packet rate. The averaged values have been plotted in Fig. 4.6, which shows variance in network transit time for each packet rate.

The other factors in the overall speaker-to-listener delay, in addition to the relative network transit time, are packet assembly time and minimum transit time. The former is easily calculated from the packet size: 10 ms times the number of parcels in the packet. The latter is that of the fastest packet (it is assumed here that it is approximately the same for all packet rates). The justification of this assumption is twofold: First, since each transmission includes periods of both speech and silence, some packets from the beginning of speech periods in each transmission should travel through the network after it has become quiescent during a silence interval. Second, the most critical resource in the ARPANET appears to be packet buffers in the packet switches, and each packet is allocated a full-size buffer regardless of the actual number of bits contained in the packet. The difference in the amount of time required to serially transmit a packet over the communication lines for different size packets is small compared to the processing time required in the packet switches.

A previous study [2] has shown that a typical minimum transit time on the ARPANET from Lincoln to ISI is 250 ms. Using this value as the minimum network transit time, Fig. 4.7 shows the sum of packet assembly time and overall effective network transit time required to properly receive 100%, 90% and 80% of the speech packets "in time" at each of the five packet rates. The optimal packet rate that minimizes the overall speaker-to-listener delay is in the range of 10 packets per second. The 90% and 80% lines are included to give a better idea of the shape of the curve; they are not recommended as acceptable reproduction levels.

Conclusion. This experiment determined the optimal packet rate for speech transmission at the given data rate for one pair of sites on the ARPANET at the time the experiments were conducted. These qualifications are necessary to emphasize that there is no single optimum value. Network performance between two closer sites is better than that measured here. Furthermore, modifications to improve speech transmission have been made to the ARPANET packet switches since this experiment was conducted. However, the overall speaker-to-listener delay increases much more rapidly for packet rates above the optimum value than for packet rates below the optimum value. Therefore this experiment showed that a good static choice of packet rate can be made that will accommodate most network conditions.

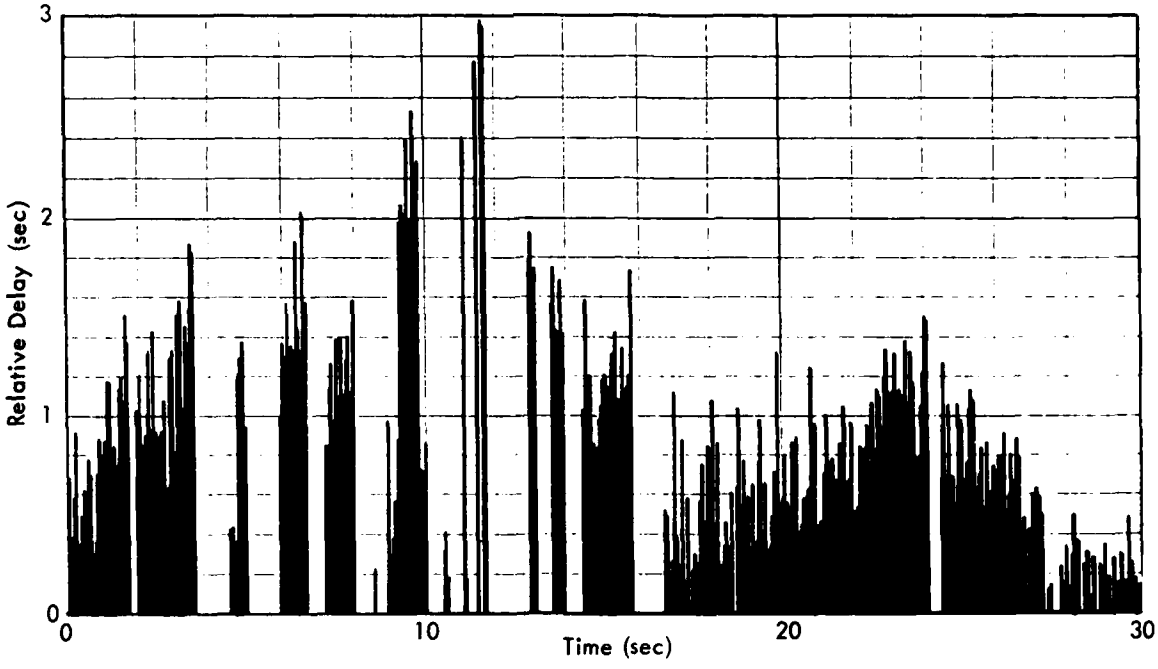


Figure 4.2 Network transit time relative to minimum
(5 parcels/packet, 20 packets/second)

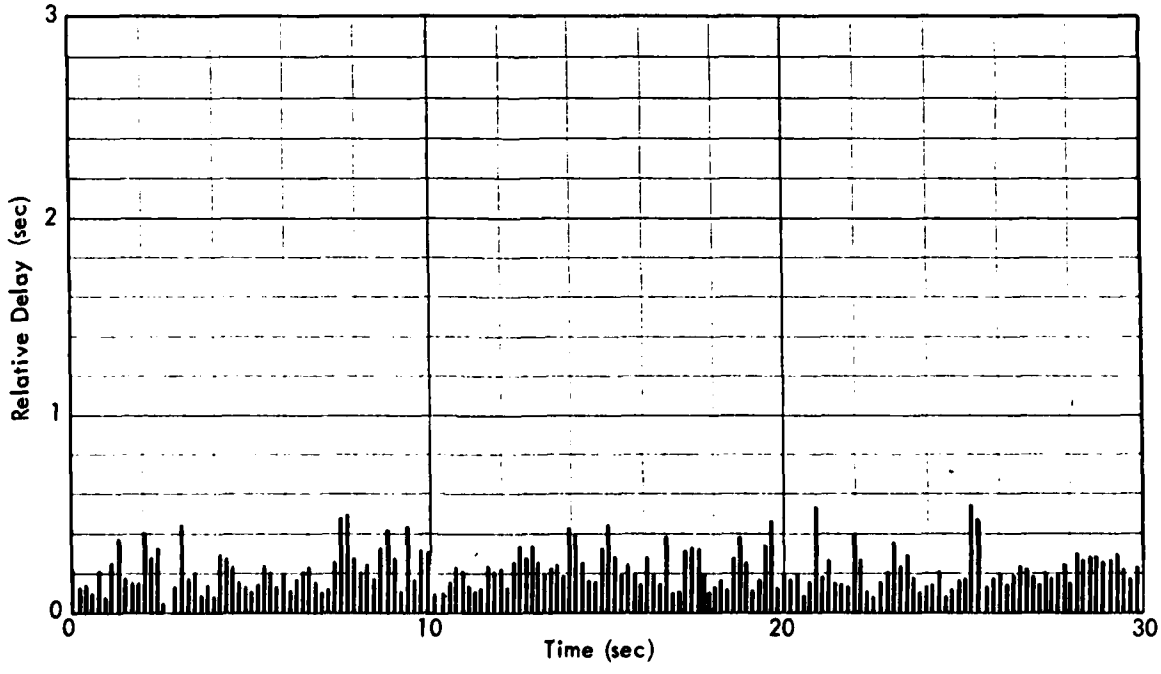


Figure 4.3 Network transit time relative to minimum
(18 parcels/packet, 5.6 packets/second)

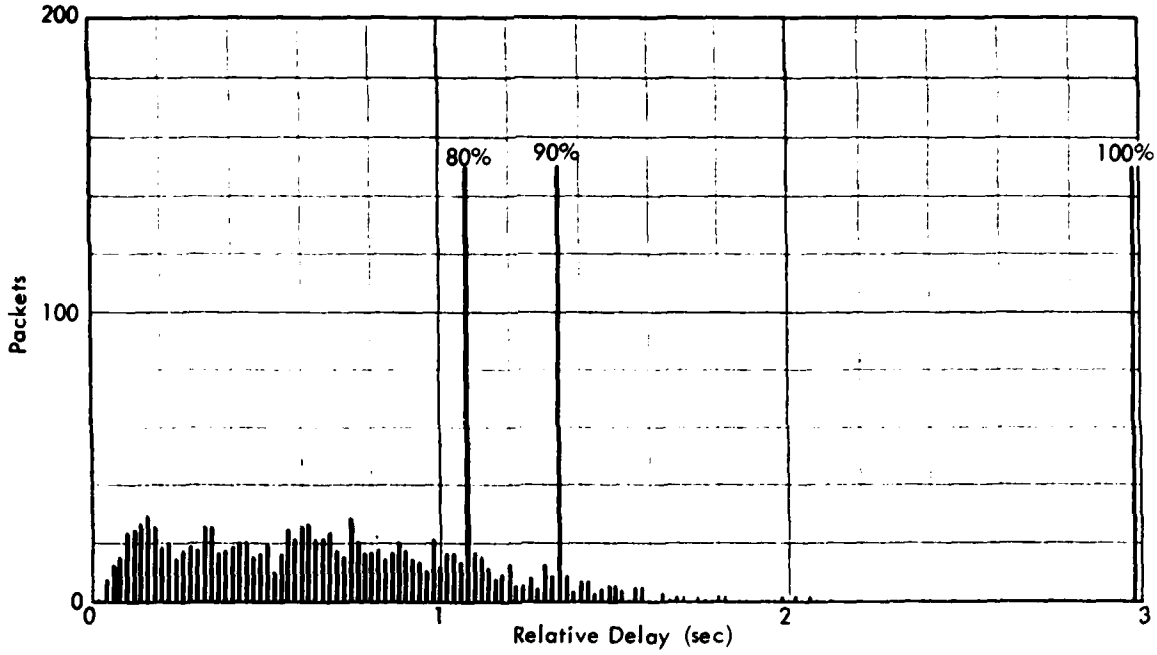


Figure 4.4 Histogram of relative transit times
(5 parcels/packet, 20 packets/second)

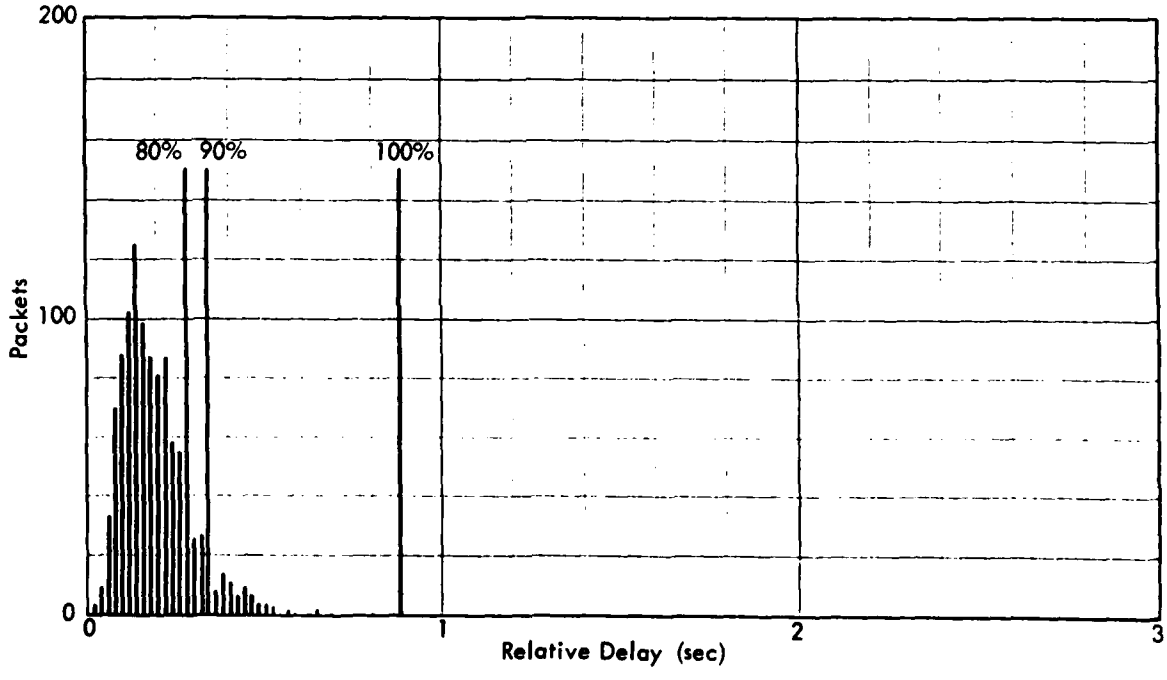


Figure 4.5 Histogram of relative transit times
(18 parcels/packet, 5.6 packets/second)

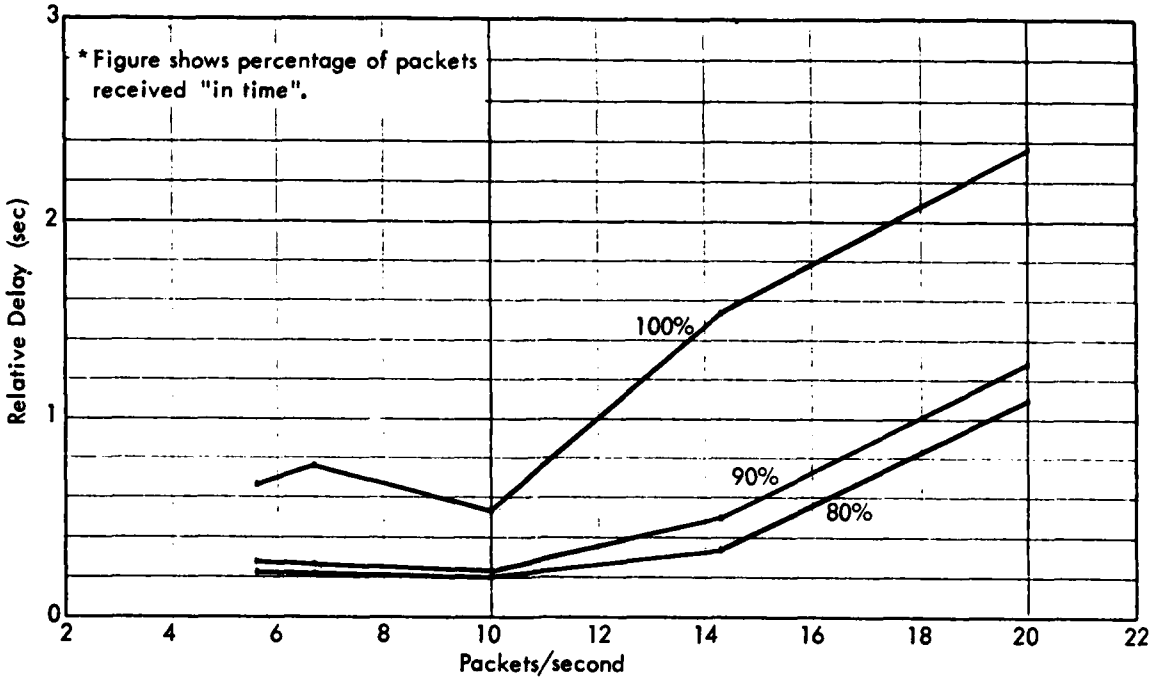


Figure 4.6 Relative delay versus packet rate*

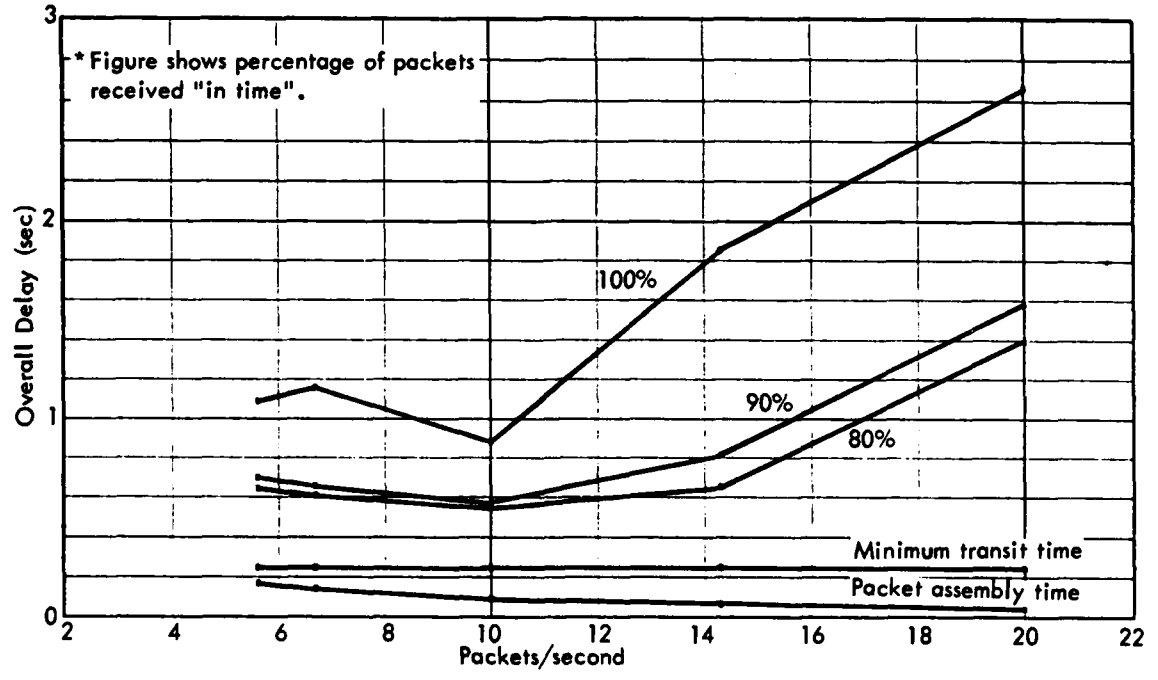


Figure 4.7 Overall delay versus packet rate*

"IMP Blocking" Tests

The third experiment was conducted to test a hypothesized bottleneck in packet voice transmission. It was thought that a bottleneck might occur at the output of the IMP at the packet voice source. Each ARPANET IMP has eight "logical channels" that correspond to buffers inside the IMP. However, in order to minimize the possibility that relatively high rate packet voice might flood the network and degrade its performance, the type of message used by packet voice (type 0, subtype 3) was restricted to using no more than two of the eight logical channels. It was thought that this restriction might be seriously affecting the performance of packet voice systems, causing longer delays and more out-of-order and lost packets.

Therefore a test was run by ISI with the cooperation of Lincoln Laboratory and the Network Control Center (NCC) at BBN. The test was run on different days, and consisted of the transmission of packet voice segments of about a minute using a very high packet rate from Lincoln to ISI. The number of allowable logical channels for packet voice was varied by NCC between one out of eight to eight out of eight, by patching the code in all ARPANET IMPs during the period between packet voice transmissions.

It should be noted that ISI used the cooperation of Lincoln during these experiments rather than the PSMF because measurement of the relative arrival time of each voice packet was desired rather than the average and other statistical measurements then implemented at the PSMF.

The first blocking test showed a dramatic decrease in the number of out-of-order packets when four logical channels were available rather than two, with little further improvement noted if six or all eight were available. The second blocking test was not as dramatic, but did show an improvement. The NCC watched both tests closely and noted no ill effects on other network traffic.

After the tests were analyzed, the NCC changed the allowable number of logical channels for packet voice from two to eight, and no effect on other traffic has been observed.

SOFTWARE

The development of new systems support software has decreased over the last few years, and only a small amount was done last year, primarily on a UNIX-compatible file system for EPOS, which is still in progress. Most software work has been in the support of the various speech applications.

Some of ISI's software effort went into helping bring up at BBN a real-time packet voice system based on a PDP11/40 and FPS AP-120B. The BBN installation uses the ISI-developed EPOS operating system and the ISI FPS LPC implementation, with minor modifications. Some minor difficulty resulted from the fact that the hardware systems are not identical: ISI uses a PDP11/45 instead of a PDP11/40, and the analog to digital (A/D) and digital to analog (D/A) converters at BBN are not located on the FPS, as they are at ISI. Much of the difficulty was caused by differences in the TENEX support software, such as linkers, compilers, etc., used by BBN and those at ISI.

The BBN system is currently up and running, and will be a reliable, readily available tool to help ISI and others in packet speech development on the ARPANET.

HARDWARE

The primary hardware-related development has been a move into a new speech/graphics lab and computer room area. The new speech lab is somewhat larger than the previous lab and has a raised floor, which makes cable routing easier and neater and cable connections more reliable. The NSC PDP11/45, FPS AP-120B, A/D and D/A converters, and associated hardware are located in a raised-floor computer room adjacent to the speech/graphics lab. The new lab will be a significantly better facility for both research and demonstrations. The co-location with the graphics lab will be very useful for future efforts in integrated multimedia (i.e., speech and graphics) communications.

In an effort to allow more flexible experimentation with relatively wide-band CVSD voice data, the four-terminal CVSD vocoder hardware has been moved from the PDP11/45 to an LSI-11, which is connected to the PDP11/45 via a DMA interface. This relieves the PDP11/45 from handling the interrupts from the word-at-a-time CVSD vocoder interface, which occurred once per millisecond. Because of the multiprocess nature of ISI's EPOS PDP11/45 operating system, such a regular, high-rate interrupt is a large load on the PDP11/45, due to the necessary context switching. Programming of the LSI-11 is in progress.

IMPACT

As packet voice technology becomes more mature, its potential impact is becoming clearer. Although much of the original impetus for the packet voice effort was its potential value as a relatively low-cost means for secure voice communication, the inherent efficiency of integrated packet-switched data and voice networks is becoming more and more apparent.

The ARPANET packet voice experiment is already beginning to affect military plans for future secure voice communications systems, and will greatly influence plans for command and control systems, particularly when packet-switched integrated multimedia communications techniques are developed.

At long last the efforts by the ARPA NSC group and others working on high-quality low-bandwidth speech communications are beginning to have an effect on the consumer market. Texas Instruments has introduced a low cost ($50) consumer product using a single-chip LPC synthesizer. The intense competition in the consumer market is sure to bring down the price of narrowband vocoders, and thus allow narrowband packet voice communications at a much lower cost.

REFERENCES

[1] *Final Report for the Project: Economic Analysis of Integrated DoD Voice and Data Networks*, Network Analysis Corporation, Contract DAHC 15-73-C-0135, ARPA Order 2286.

[2] Forgie, J. W. and Mc Elwain, C. K., *ARPANET Delay Measurements*, NSC Note 70, M.I.T. Lincoln Laboratory, July, 1975.

5. COMMAND AND CONTROL GRAPHICS

Research Staff:
Richard L. Bisbey II
Dennis Hollingworth
Gertrud Mellstrom
Elaine Thomas Sonderegger

Consultant:
Danny Cohen

Research Assistant:
Richard Shiffman

Support Staff:
Debe Hays

THE PROBLEM BEING SOLVED

Effective command and control depends on accurate and timely exchange of information between command levels and effective presentation of relevant data. Communication both up and down the command chain is required, as well as communication with data bases, either static or dynamically updated. Advanced information processing methods can and should be exploited to manage and augment information exchange, thereby making it easier for military personnel to interpret and respond to events.

Effective communication is crucial for real-time operation in crisis, as the 55 hours of voice conferencing during the Mayaguez incident testify. Simultaneous graphic communication, typically two-dimensional information like maps and charts, can greatly enhance both the quality and the efficiency of communications, and provide depth to the information exchange. Furthermore, graphics are probably the most effective and universal way of communicating spatial information.

ISI is developing for ARPA a computer-based graphics system for use in a command and control environment. Early versions of the system are currently in use at the Naval Ocean System Center (NOSC) as part of the Navy's Advanced Command and Control Architectural Testbed (ACCAT). ISI is also developing graphic application programs, including a Situation Display, which is force information displayed on a geographic background, that demonstrate use of the graphics system.

GOALS AND APPROACH

The principal aim of the effort is the development of a display-device-independent vector graphics system. "Display-device-independent" means that graphic application programs can be written without regard to the particular display device on which the output will ultimately be displayed. The system being

developed achieves display device independence by providing the application program with a set of application-independent, device-independent, two-dimensional vector graphic primitives (Graphics Language [1]) by which pictures can be described and interacted with at the application level.

The particular graphics model used is based on structuring pictures as sets of subpictures which are absolute-transformed-segments, as defined by Newman and Sproull [2]. At the most general level, the application program deals with graphics in terms of named picture elements, called segments. Segments can be created, destroyed, merged, made visible or invisible, made touch-sensitive, and highlighted. Segments themselves are specified in terms of generic graphic primitives including vectors, arcs, dots, text, and filled sectors and polygons, that may vary according to color, intensity, type-face, and line-type.

Graphics Language (GL) defines the set of application-independent functions for performing the above operations. Since several programming environments are expected to be used in the command and control environment (mainly FORTRAN, but also LISP, BLISS, MACRO and other languages) GL is defined (and implemented) in such a way that it can be easily used by application programs written in any of them.

The binding of the application program to a particular display device is deferred until program execution time, when the generic graphics primitives are mapped by the system into specific operations and display modes appropriate to the device selected. The quality of the displayed picture resulting from user application requests is limited only by the capability of the display device.

The second significant aim is that the graphics system be distributable across multiple host computers interconnected by a communications network (such as the ARPANET). This allows the graphics display device to be located away from the graphics application program; it also allows the computational load introduced by the graphics system to be distributed across multiple processors, balanced to the computational resources and communications bandwidth available. The graphics system achieves distributability by virtue of its modular design and implementation, i.e., the system consists of a series of isolatable functions that communicate via a common communications mechanism (see [5] for more details).

The design separates the issue of system functionality from communications. In other words, the functionality of the system does not depend on the physical processor on which a given function resides during a graphics session. The physical location of graphic functions (as well as the display device) affect performance issues only; they have no effect whatsoever on the application program. Moreover, any intraprocess/interprocessor communications mechanism

might be used for communications between functions of the graphics system (including telephone, radio, or digital network/internetwork links). As suggested by R.F. Sproull in "Network Graphics Isn't Networking" [3], a graphics system used on a network does not have to address networking issues, but only device independence.

Modular organization of the graphics system has several benefits in addition to distributability. It allows the system to be easily extended to support other application languages and display device types. The former is accomplished by replacing an application interface component by one which interfaces to the new language, and the latter by replacing the component that generates device orders. Such replacement has no effect whatsoever on the application programs or the fashion in which application programs use the graphics system and graphics language.

Modularity also allows new graphic functions to be easily added to the capabilities of the basic system. The architecture supports a single-terminal-to-single-application graphics environment. Additional graphic functions may be added to support more complicated configurations, including multiple programs concurrently connected to a single terminal and a single application program simultaneously generating graphics output for and accepting input from multiple (possibly dissimilar) display devices.

PROGRESS

Graphics System

In January 1977, ISI delivered a nondistributable (Level 1) version of the graphics system. This initial version supported the Tektronix 40XX series display and was later extended to support the Genisco GCT-3000 raster-scan, bit-map color graphics display. Extensions were made to the Level 1 system during this reporting period to support an additional application language, several additional graphics display device types, and an additional connection protocol.

Three documents were written describing the graphics language [1], the internal graphics protocol [4], and the general system architecture [5]. Work began on implementation of the distributable (Level 2) version of the graphics system.

Situation Display

Concurrent with the development of the graphics system, ISI is developing several command and control applications to demonstrate various capabilities of the graphics system. One of these is Situation Display, a natural language C2 information retrieval system with graphic output. Situation Display is a cooperative effort between ISI and SRI International. The natural language and data retrieval capability is provided by SRI's LADDER (Language Access to Distributed Data with Error Recovery) [6] while the graphics presentation is controlled by ISI's SDGS (Situation Display Graphics Subsystem--which in turn makes use of the graphics system developed at ISI). The first version of Situation Display was delivered to NOSC in June 1977. During this reporting period, Situation Display was extended to provide additional display capability including highlighting, selective display and amplification of ship information, and locally retained maps. The facing page shows examples of Situation Display output. A document was written describing the system architecture [7].

IMPACT

The principal impact of this work is in developing a graphics system architecture that accommodates system decentralization and distributed graphics. Such a system architecture will facilitate graphics/user environments of widely varied display capabilities, storage capabilities, and processing capabilities. An example of such a system is a ship-based graphics system that must interact with and possibly supplement one or more land-based graphics systems associated with large computer resources. The graphics system is intended to provide a sufficiently rich graphics capability to support a wide variety of applications and terminal types.

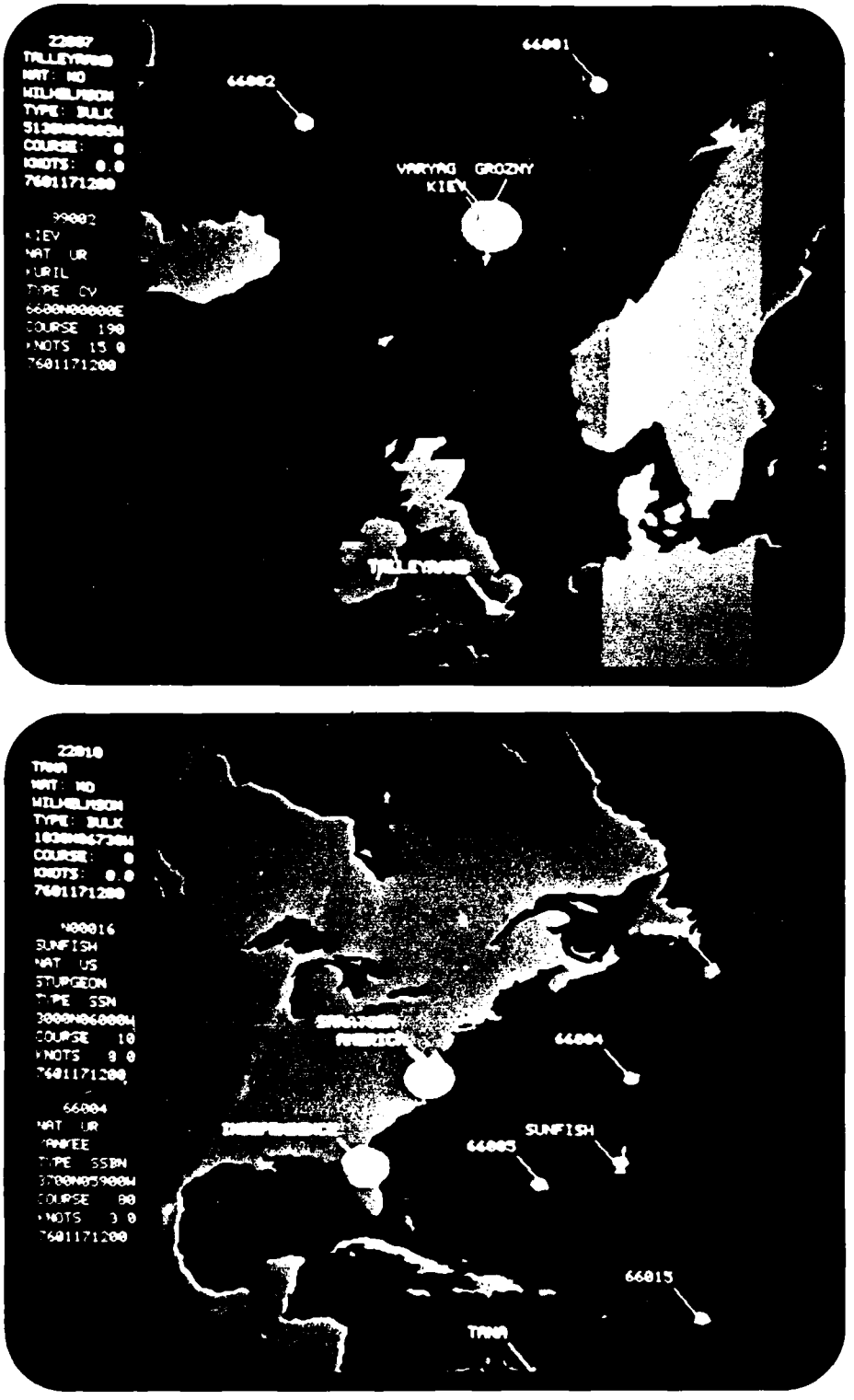


Figure 1. Typical situation display outputs using Graphics System.

REFERENCES

[1] Graphics Language, USC/Information Sciences Institute, December 1977.

[2] Newman, W.M., and Sproull, R.F., *Principles of Interactive Computer Graphics*, McGraw-Hill Book Co., Inc., New York, 1973.

[3] Sproull, R.F., "Network Graphics Isn't Networking," *Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks*, May 25-26, 1976, Lawrence Berkeley Laboratory, University of California.

[4] Graphics Protocol, Information Sciences Institute, December 1977.

[5] Bisbey II, R., Hollingworth, D., A Distributable, Device-Independent Vector Graphics System for the Military Command and Control Environment, USC/Information Sciences Institute, April 1978.

[6] Sacerdoti, E., *Mechanical Intelligence: Research and Applications*, Final Technical Report April 12, 1976-October 9, 1977, SRI International, December 1977.

[7] Bisbey II, R., Hollingworth, D., Situation Display: A Command and Control Graphics Application, USC/Information Sciences Institute, April 1978.

6. AUTOPSY: SYSTEM FOR (SEMI-) AUTOMATIC TRANSLATION OF OLD PROGRAMS

Research Staff:
Stephen D. Crocker
Peter Alfvin

Research Assistant:
Jeff Cook

Consultants:
David Martin
Hanan Samet

Support Staff:
Pamela Kaine

MOTIVATION

The Defense Department will soon adopt a new programming language, Ada, which will be the standard language for embedded computer systems. This new standard is expected to help reduce the cost of development and to improve the reliability of the resultant software. The new language design will be accompanied by the development of new tools to aid programmers in using it. In addition to the obviously necessary compilers for each machine, we can also expect specialized editors, debugging systems, verification systems, and the like. In short, the language will be supported far better than any language has been to date.

One of the problems that has not been addressed during the development of the new language is what to do with all of the programs written in the old languages, such as the Navy's CMS-2, the Air Force's JOVIAL and the Army's TACPOL, among many others. Many of the programs written in these languages can remain as they are and slowly fade from use. Many others, however, will need to be updated to meet changing requirements or to run on new hardware.

While it may be possible to maintain these programs in their present form, much would be gained if they could be translated into Ada. As mentioned above, the support mechanisms for Ada are likely to be far superior to those now in use for existing languages. Moreover, for new computers, it is unlikely that compilers for old languages will exist at all.

This project addresses the problem of program translation.

OVERVIEW

The most straightforward idea for translating programs is to build a complete transducer that reads programs in the old language and translates each construct or sets of constructs in the old language into one or more corresponding constructs in the new language. While this approach may work in some cases, it often fails when the new language is not a complete superset of the old language. In the present case, we do not know the precise definition of Ada, but we know enough to be sure that there is no real possibility of translating into Ada arbitrary programs written in the several old languages.

The Autopsy project is a research effort that combines automatic translation techniques with an interactive system to provide the human manager complete control over the translation process. The human may ask the system to attempt translation of all or some of an old program. The human may also control the translation more intimately by specifying that a particular transformation be applied or by supplying the replacement code himself.

The proposed system will evolve to become more and more automatic, but one essential aspect of the system will be fixed at the beginning: the complete history of the translation of an old program into a new one will be recorded and be accessible for inspection and/or modification at any time.

This research is being carried out in three major epochs. Epoch one is devoted to the development of a basic system for translating from one old language to one new language. Epoch two will be devoted to a sophisticated version of the same system. Epoch three will be devoted to the automatic development of translation systems.

The primary purpose of the epoch one system is to demonstrate the utility of our approach and to provide a firm design for the development of a sophisticated system. Only two fundamental mechanisms are being included in this basic system. Algorithms for automatically translating constructs in the old language into corresponding constructs in the new language will be written wherever possible. In the absence of automatic algorithms, the user will have access to an editor to simply write the replacement code in the new language. As noted above, however, an early component of this system will be a complete recording facility. The complete history of the transformation of a program in the old language into one in the new language will always be available.

The epoch two system is intended for use by a small, selected set of programmers in a military laboratory. In addition to the capabilities of the epoch one system, the epoch two system will have facilities for pattern-directed

transformations to be invoked under a user's control and for formal verification of some types of replacement of code. By this time, the system will be documented and easy to use. Prettyprinting routines for the code in the new language and other bookkeeping aids will be provided to make it easy for the user to understand the structure of programs in the new language and how the new program is related to the old. Of course, the system is not designed to invent algorithms or provide clean code in the new language when the old code was convoluted. But the system will attempt to retain what structure is available and provide some tools for cleaning up the code by hand. We envision that a serious test of the epoch two system should take place during the third research year, with support from our staff. During that year, we will also refine the system and enhance its ability to implement context-sensitive transformations.

Our long-term goal is to automate the production of translation systems. Using the epoch two system as a model, we will examine the algorithms developed by hand and see what it would take to produce these algorithms automatically, given formal descriptions of the old and new languages. We expect to apply whatever techniques we develop towards the production of a translation system for an additional input language, i.e., one different from the language input to the epoch one and two systems.

SYSTEM STRUCTURE

The translation system is based on the concept of a common internal representation of programs. Old source programs will be translated into this format and new source programs generated from it. All analysis and transformations, whether system- or user-directed, will be performed on this internal representation. In addition to providing a rich operating environment, this approach allows us to present a uniform description of program translation for any pair of old and new languages supported by the system.

Choosing the internal representation is obviously a critical task. If the old and new languages are quite different (e.g., FORTRAN and LISP)the task might be close to impossible. However, we expect that this system will be used only when the old and new languages are closely related. In such cases, we expect to be able to find quite reasonable internal intermediate representations.

The system will be composed of three subsystems (Figure 6.1). The first subsystem will be responsible for inputting the old source programs and translating them into the internal format. The second subsystem will perform any necessary transformation on the program. And the third subsystem will output the new source programs. Each of these subsystems will be subject to increasing degrees of automation as the project progreses.

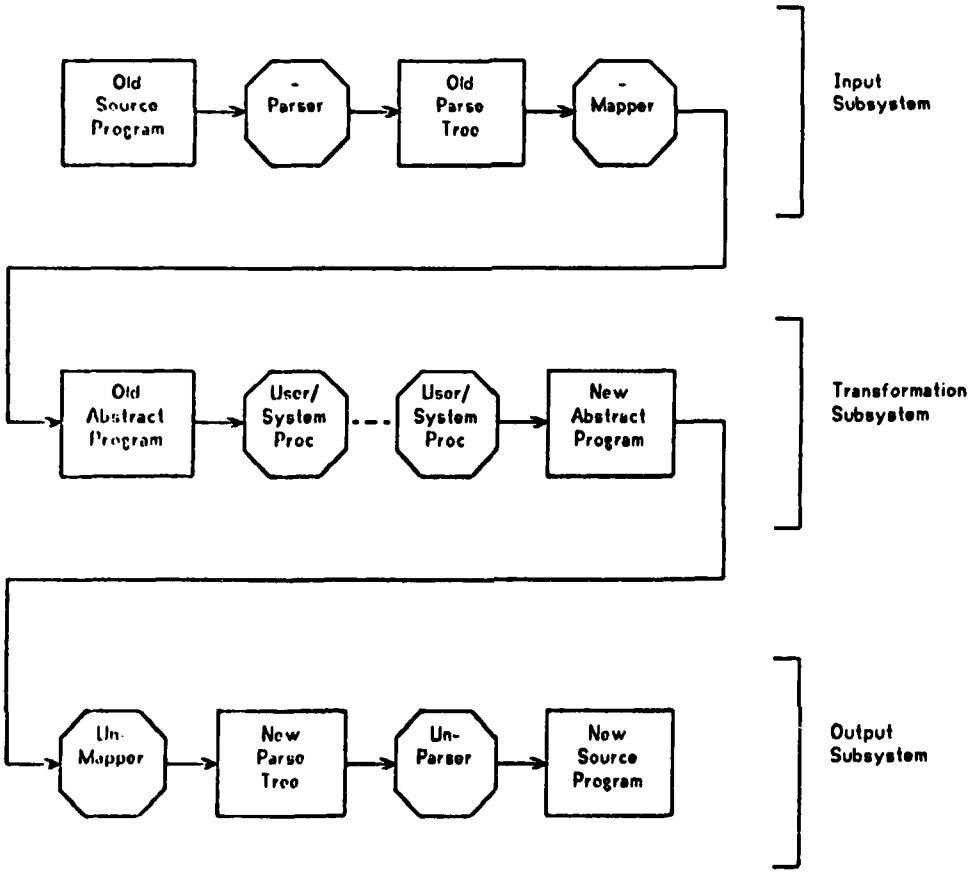


Figure 6.1

Input Subsystem

This subsystem consists of two processes, shown as octagons. The first process, the Parser, reads the old source program and generates a parse tree. This parse tree will be uniquely determined (assuming no syntactic ambiguity) by the BNF of the source language. The derived parse tree is then translated into an abstract form.

The terms "abstract form," "abstract syntax," and "abstract program" have received widespread use, with some resultant confusion. When we speak of the "abstract form" of a program, we are referring to that internal format which most conveniently represents that program, where "convenience" is determined by the operations we need to perform. While this format will be similar to the derived parse tree, the differences are significant enough to merit a distinction. For instance:

> In the BNF for most concrete languages, an operator precedence is defined through a hierarchical set of production rules. This hierarchy results in a great deal of complexity in the parse tree representation of expressions involving these operators. A simple integer might be represented as an instance of an expression, which was a sum, which was a product, which was a term, which was (finally) the integer itself. Once a program has been parsed, this structure is no longer useful. Expressions can be represented in a straightforward prefix notation, requiring no additional structure.
>
> Another common feature of concrete languages is that they frequently allow defaults. In the case of an iterative loop construct, such as "for i from j to k by m do s," the "increment" clause ("by m" in this example) is often optional, in which case an increment of (say) 1 is assumed. In order that the semantic processing of these loops might be somewhat simplified, the default increment would be provided in the abstract form.
>
> Another reason for utilizing an abstract form of the source program is to provide a normalized representation for multiple source languages. It is assumed that the source languages will have many features in common. In order that equivalent constructs in old languages need not be individually treated, a common representation for these constructs must be defined. The abstract form, then, would support the union of the constructs supported by the old source languages.
>
> Finally, in order that the transformations on the source languages be performed within a single representation, the abstract form must be expanded to include any constructs in the new source languages which are to be utilized in the transformation process. This will simplify the system by allowing us to separate the problem of generating the appropriate new language constructs from the problem of generating the external representation of those constructs.

An example of the above processing is given in Figure 6.2 on the following page.

```
1.  <Exp> := <Factor>
2.  <Exp> := <Factor> <AddOp> <Exp>
3.  <AddOp> := +
4.  <AddOp> := -
5.  <Factor> := <Term>
6.  <Factor> := <Term> <MultOp> <Factor>
7.  <MultOp> := *
8.  <MultOp> := /
9.  <Term> := <Variable>
10. <Term> := <Integer>
11. <Term> := ( <Exp> )
12. <Assign> := <Variable>  <Exp>
```

Parse Tree for "AB+A*6"

```
(P12 A (P2 (P5 (P9 B))
           (P3)
           (P1 (P6 (P9 A)
                   (P7)
                   (P5 (P10 6))))))
```

Abstract Form

```
(Assign A (Add B (Multiply A 6)))
```

Figure 6.2. Sample BNF for Assignment Statement

Transformation Subsystem

The purpose of this subsystem is to translate the abstract program into a form that can be represented in the new source language. More specifically, all constructs in the abstract program that are not available in the new language must be translated into equivalent available constructs. In addition, the user may want to perform an otherwise unnecessary transformation for the purpose of readability, speed, or other considerations. While the organization of processes within this subsystem is not clear at this time, certain ideas stand out as being critical:

The system should support both user-supplied and system-supplied translations. As the system evolves, the system can take on more and more

of the burden of translation, but there will always be a need to support user modifications.

User translations may take the form of editing commands or the application of user-generated procedures to the abstract program. In each case, the user's input will be recorded and "back-up" facilities will be provided.

System translations will consist of a library of translations organized by language construct. That is, each "offending" construct in the old language might have one or more interpretations in terms of constructs in the new language. Many times these interpretations will be partial, and success in translation will depend on the individual program. If translation depends on certain assumptions about the behavior of a particular program, the user could be queried and his answer recorded.

The history of user and system translations should be recorded in such a way that it can be reused as input to the system. In addition, an easily readable description of the translation should be obtainable from this record and a copy of the program.

For editing and documentation purposes, a concrete form of the abstract language will be provided.

Output Subsystem

This subsystem is the converse of the input subsystem. The abstract program is translated into a parse tree for the new source language. The reverse of the operations discussed above for the input subsystem must be performed. Then the parse tree is translated, on the basis of the BNF, into the new source language. There are some additional issues involved in this unparsing process involving spacing, comments, etc., but these are not felt to be major issues.

STRATEGY FOR EPOCH 1

We intend to build our system from within the INTERLISP environment, making extensive use of available INTERLISP tools whenever possible. Not only does INTERLISP provide the list processing capabilities appropriate for this task, but it also gives us a firm foundation from which we can generate a customized system. In addition, some related work in this area has been carried out within INTERLISP resulting in an auxiliary collection of tools from which we can select.

A critical component of the system will be the intermediate, abstract form. The obvious LISP datatype in which to represent this form is the s-expression. However, s-expressions are unwieldly from a user's point of view. Fortunately, INTERLISP provides a facility (CLISP/DWIM) which allows for table-driven transformations from s-expressions into more readable forms (e.g., prefix-infix transformation). Consequently, we will be able to define a tractable representation of our intermediate language for the user to work with while retaining the desired internal format. This external representation can also be be used for documentation purposes, so that any program transformations can be expressed at the level at which they were performed (the abstract level) rather than having to translate the transformations into operations on either the old or new languages.

The processes required in the input subsystem can be easily developed from existing tools. A program has been written [Wile] to generate INTERLISP parsers from a BNF-like input specification. Though some modifications to the program may be required for production use, it does provide us with a means of quickly generating parsers for arbitrary input languages. The INTERLISP input/output facilities allow straightforward specification of lexical analyzers for use with these parsers. In addition, the extended BNF input notation provides a means of specifying translations to the parse tree during the parsing subsystem. Making use of this mechanism, we can implement the post-parse translations in a straightforward and efficient manner. The above method of lexical analysis, parsing, and translation has already been used successfully in our work with ISPS.

The transformations at this point will be either hand-coded system functions or user-supplied commands to the editor. The system functions will hopefully be able to translate fairly completely most of the constructs in the old language, but there will undoubtedly be some shortcomings. No general library of transformations will be supplied.

Another responsibility of the transformation subsystem is to record the user's actions. The INTERLISP "history list" feature provides an initial model for this process. Related to the history list is the INTERLISP "programmer's assistant" which supports the concept of allowing the user to inspect, retry or undo the effects of previous commands. This safe and flexible environment encourages experimentation with the system and generally increases productivity. Another LISP feature, TRANSOR, is concerned with performing translations from one LISP dialect to another. It provides a facility to specify transformations as a series of commands to the INTERLISP editor on a construct-by-construct basis. In addition to the transformed expression, it generates a list of unresolved program areas. This process is identical to the process we are attempting to support.

The processes in the output subsystem are concerned with the generative use of grammars rather than their use for parsing purposes. Because we know of no available tools similar to the parser generator mentioned for the input subsystem, we will have to either create such a tool or write the processes by hand. The main issues here revolve around the loss of information generally associated with lexical analyzers. Program comments, which are usually discarded during parsing, must be retained. Program spacing and indentation will have to be regenerated. This process corresponds to the INTERLISP "prettyprint" feature and is a reasonable approach to take.

STRATEGY FOR EPOCH 2

The key issues addressed in this epoch are user-directed transformations and formal verification. Source-to-source transformation issues have been studied by several authors [Balzer, Standish2]. Balzer is concerned with generating efficient implementations from abstract representations of programs through successive application of catalogued transformations. Standish has published a collection of transformations [Standish1] and implemented an interactive production system. We are concerned, at least initially, with replacing those specific constructs in the abstract program not supported in the new source language. As such, our problems are a subset of those addressed by the previous work and we should be able to make use of the transformations and transformation techniques already developed. It may also turn out that general transformations are required in order to resolve space and efficiency considerations, in which case the previous work would be more fully applicable.

Formal verification techniques would be utilized by the user in justifying the translations or substitutions he supplies himself. The verification proof would become a part of the record of the user's activities and would be required to conform to the same rules regarding reusability and documentation as the user's translations.

Verification is a very complex problem, of course, and we do not intend to guarantee that we can develop tools to support verification of arbitrary replacement of one program by another. However, recent work by Samet [Samet1, Samet2, Samet3] suggests that it is possible to verify the equivalence of two programs automatically if the two programs use equivalent data structures and progress through equivalent control paths. Samet's techniques would not apply, for example, to showing that a quicksort algorithm is equivalent to a bubble sort. However, in the program translation problem, the primary goal is to convert the same algorithm to a new language. Changing the basic data structures or algorithms is really a separate problem. We recognize, of course, that there is

strong motivation to improve a program while it is being converted, and we expect that our proposed system will provide a useful general framework for recording the changes and cataloging the versions of the program, but we have no plans for supporting verification of these improvements beyond the simple conversion of algorithms from the old language into the new.

The system will be documented for use by selected outside experts.

STRATEGY FOR EPOCH 3

Eventually, we would like to automate the generation of translation systems, based on formal definitions of the old and new languages. There is no accepted best method of formal definition of a programming language, but some experience has been accumulated. The syntax definition is fairly easy: some form of BNF is almost always used. The semantic definition is much more complex. We have been experimenting with the use of attribute grammars to formally specify ISPS--a machine description language--and have been achieving moderate success. We are not yet ready to take a firm position on our preferred method of language definition, but consider this to be one of the research issues to be addressed. To be clear what the bounds of the research problem are, we summarize the desired result:

> We need a formalism for describing the syntax and semantics of a programming language. This formalism must be adequate for describing Ada and the most useful of the old languages--probably CMS-2 and TACPOL. The formalism must be readable by humans and present the information about the language in a reasonably intuitive format. The formalism must also be readable by a machine and should be amenable to the translator generator. To keep the problem within bounds, the formalism need not handle every perverse kind of programming language. If need be, the formalism might also put a boundary around particularly difficult constructs in otherwise tractable languages.

Given some means of formally specifying the syntax and semantics of the programming languages, we need some means of generating a translator. Two means are available. One is to build an automatic translator generator. The other is to provide a means for the user to specify the translator. We intend to pursue both paths. A translator generator will be attempted which examines the definitions of the old and new programming languages and looks for an algorithm for translating constructs in the old language into constructs in the new language. In general, this translator generator will have only partial success and will leave part of the task to the user.

The user will then have the option of specifying how to translate some of the constructs in the old language into constructs in the new language. Of course, some assurance is desired that the translation proposed by the user preserves the functionality of the program, and we expect to provide some means of testing the appropriateness of the user-supplied scheme.

Just as there may be some question about the adequacy of the user-supplied translation schemes, there may also be question about the system-supplied scheme. Partly to provide high visibility to the heuristics employed and partly to provide an easy path to their improvement, we expect that the translator generator will be written in a specialized and readable language that the user will have direct access to. Moreover, the translation schemes proposed by the system will be subject to the same type of testing and verification as the user-supplied translation schemes.

PRESENT STATUS

An extended BNF grammar for CMS-2M, derived from the grammar in the AN/UYK-20 CMS-2M compiler specification, has been written and input to Wile's parser generator program GOP [Wile], to produce a CMS-2M parser. The parse record corresponding to a CMS-2M program input to this parser is suitable for syntax-directed semantic computations or translations of CMS-2M programs. In addition, realizing that a conventional context-free syntactic description (rather than extended BNF) of CMS-2M may be better suited to general syntax-directed semantic specification schemes such as attribute grammars [Knuth], a program for converting extended BNF grammars to equivalent context-free grammars has been written.

Attribute grammars (AGs) [Knuth] are a useful and powerful means for specification of machine-implementable syntax-directed semantics. An AG consists of an underlying context-free syntax specification and an accompanying semantic specification keyed to the syntax. A choice of semantic domain, i.e., the objects comprising the semantic definitions, is inherent in this semantic definition. AGs can be used to define programming language translators or more abstract semantic definitions such as denotational semantics [Stoy].

A comprehensive system for the definition and machine implementation of AGs has been designed and partially implemented. Those portions of the system that have been implemented are an input processor and a circularity tester. The circularity tester is necessary to determine that the set of attribute equations keyed to the syntax of a language do not define a circular, and hence unsatisfiable, set of computations.

Algorithms have been developed, but not implemented, for attribute evaluation and conversion of AGs to purely synthesized form. The algorithm for automatically converting an AG specification into a simple parse-tree-traversing pushdown automaton for evaluating the attributes on a given parse tree of that AG has been designed and proven correct. The algorithm to convert an AG to purely synthesized form is based on a symbolic-execution variant of the attribute evaluation algorithm. It converts the original AG into an equivalent one which contains only synthesized attributes. Synthesized attributes are ones computed only from lower nodes and are distinct from inherited attributes which are computed from adjacent or higher nodes. A purely synthesized AG is trivially non-circular, hence the conversion process requires a non-circular AG as input.

REFERENCES

[Balzer] Balzer, R., N. Goldman, and D. Wile, "On the Transformational Implementation Approach to Programming," *2nd International Software Engineering Conference*, October 1976, San Francisco, pp. 337.

[Knuth] Knuth, D. E., "Semantics of Context-Free Languages," *Math Systems Theory 2*: 1968, 127-145.

[Samet1] Samet, H., Automatically Proving the Correctness of Translations Involving Optimized Code, Ph.D. Thesis, Stanford Artificial Intelligence Project Memo AIM-259, Computer Science Department, Stanford University, 1975.

[Samet2] Samet, H., "Compiler Testing via Symbolic Interpretation," *Proceedings of the ACM 29th Annual Conference*, 1976, pp. 492-497.

[Samet3] Samet, H., "A Normal Form for Compiler Testing," *Proceedings of the Symposium on Artificial Intelligence and Programming Languages*, 1977, pp. 155-162.

[Standish1] Standish, T., D. Harriman, D. Kibler, and J. Neighbors, *The Irvine Program Transformation Catalogue*, Computer Science Department, University of California at Irvine, January 1976.

[Standish2] Standish, T., D. Kibler, and J. Neighbors, "Improving and Refining Programs by Program Manipulation," *Proceedings of the 1976 ACM National Conference*, Houston, Texas, October 1976, pp. 509-516.

[Stoy] Stoy, J. E., *Denotational Semantics*, MIT Press, 1977.

[Wile] Wile, D., "Generator Of Parsers," Internal Memorandum, Information Sciences Institute, November 1976.

7. PROGRAMMING RESEARCH INSTRUMENT

and

MULTIPLE MICROPROCESSOR EMULATION

Research Staff:
Louis Gallenson
Steve Crocker
Peter Alfvin
Alvin Cooperband
Joel Goldberg

Research Assistants:
Peter Liao
William Morgart
Sarma Sastry

Support Staff:
Pamela Kaine
Raymond L. Mason

BACKGROUND

The PRIM (Programming Research Instrument) project, initiated in 1972, provides a flexible tool for research in computer architecture. The PRIM system is built around a high-speed microprogrammable computer used for emulation slaved to a general-purpose time-sharing system. The system is designed to accommodate many concurrent users and is also available to remote users over the ARPANET.

The initial PRIM hardware and software were operational in May 1974 with the ability for remote users to compile and debug microcode. The basic PRIM system, which emphasizes generalized software development tools (with some specific tools), became operational in March 1976.

The original PRIM system uses a Standard Computer Corporation MLP-900 for the high-speed execution. Only one copy of this hardware was ever built. Recently, a number of groups, particularly Rome Air Development Center (RADC) and Naval Ocean Systems Center (NOSC), have procured Nanodata QM-1's for emulation work. In keeping with our philosophy that such machines should be attached to general-purpose time-sharing systems, we recommended that these QM-1's be operated in a PRIM-type configuration. The result has been an extension of the PRIM development project to produce a QM-1-based PRIM system called QPRIM.

At the same time, interest in architecture research has expanded to include multiple processor systems. Of particular interest are closely coupled assemblies of substantial numbers (20 to 50) of small machines. One of the classic problems in

emulating such an assembly of machines is controlling the time lost in switching contexts from emulation of one processor to another. If the machines are only loosely coupled, it is possible to run each emulation for a fairly long time before switching contexts. However, for closely coupled systems, context-switching may be required as often as every (emulated) instruction. In examining the capabilities of the QM-1, it appears that enough control store is available to hold the microcode and working registers for a large number of machines. If so, the context switching time should be reducible to a very small number of microinstructions and the overhead for running a closely-coupled set of emulations kept to a reasonable fraction of the total execution time. An extension to the PRIM emulation system to handle multiple emulations is now under way. This part of the effort is named Multiple Microprocessor Emulation (MMPE).

SYSTEM DESCRIPTION

Detailed descriptions of the PRIM system are available in the previous annual report [1] and in reference documentation [2], [5], [6]. The system architecture is briefly described here and illustrated in Fig. 7.1. The PRIM system is built in three levels upon a base consisting of the MLP-900 microprogrammable processor and the TENEX timesharing system. First is the *operating system level* of PRIM, consisting of the hardware and software that support shared access to the MLP-900 from TENEX processes. Second is the *user level*, the PRIM system proper, which provides interactive access to a user at a terminal. And third is the *tool level*, consisting of the set of installed emulation tools, each providing its users with a complete target environment. The user level is for both the emulator developer and the target machine programmer.

The tool level of PRIM consists of the set of completed emulation tools available to target machine programmers plus an MLP-900 tool available to emulator developers. A completed emulation tool consists of an emulator (pure MLP-900 microcode), a table file describing that tool to PRIM, and a dummy TENEX process by means of which PRIM initializes everything for this target machine. The emulated machine produces better user debugging facilities and greater flexibility in system configuration than the original machine, at the expense of execution speed, while producing bit-for-bit compatible results on all levels of execution. Currently, PRIM also provides the AN/UYK-20 [4], Univac U1050 [5], and the Intel 8080 as completed emulation tools. The MLP-900 tool consists of the GPM compiler and a table file describing the MLP-900.

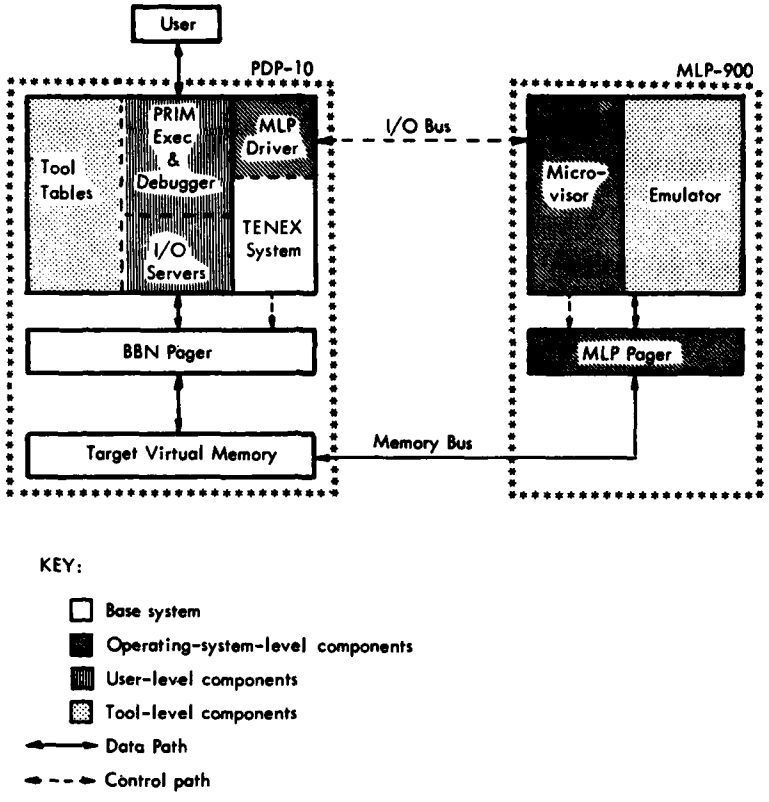


Figure 7.1 Architecture of PRIM

From the user's point of view, the PRIM system can be in one of three states: the PRIM exec, the PRIM debugger, or the emulated target machine has control. The user's initial interface to the PRIM system is with the PRIM exec. From there he can pass control either to the PRIM debugger or directly to the target machine. From the PRIM debugger, he can pass control either back to the PRIM exec or directly to the target machine. Once control is passed to the target machine, it stays there until target execution stops or is stopped by user intervention.

DOCUMENTATION

User documentation has been prepared for each of the existing PRIM tools. In user guides for the specific machine tools (UYK-20, U1050, and 8080), we assume the reader is a familiar user of the specified computer system but requires instructions for interactive code generation and debugging. For the naive

programmer a tutorial section provides detailed examples of the utility and power of the PRIM tool. A comprehensive software manual containing machine-independent information has been prepared for the general PRIM user. The *PRIM System: Tool Builder's Manual* has been prepared for users interested in writing new emulators.

CURRENT PRIM ACTIVITY (MLP-900 System)

NOSC's System Design Laboratory (SDL) is currently introducing PRIM tools to user communities within the Navy. The charter of SDL is to support the design and development of naval systems employing computers by providing a cohesive set of tools with a common user interface. The AN/UYK-20 [7] is the standard minicomputer for naval systems and the PRIM UYK-20 is an important part of SDL's capability. During FY78 the PRIM UYK-20 was used by seven different Navy development projects with an approximate total of 100 hours of MLP-900 CPU time. These figures are impressive when considering that this is the first year SDL offered this service. Based on initial user reactions to PRIM UYK-20, we expect continued growth in the user community and increased usage by each of the development projects during FY79.

In response to user request the UYK-20 tool has been enhanced to include additional I/O devices. These include: a 188C channel interface which provides an asynchronous serial data path, a generalized NTDS interface to allow users to add their own versions of emulated I/O devices; and the addition of Baudot code devices (TTY). It appears that Navy systems utilizing the UYK-20 are primarily concerned with unique I/O devices and the demand for new device emulations will continue. This requirement is satisfied with minimum implementation effort and is being performed as part of the maintenance task. The PRIM project is essentially completed and ISI will continue to be responsible for the system maintenance.

The PRIM project is also interacting with a research group at the Dahlgren Laboratory of the Naval Surface Weapons Center (NSWC) in developing communication protocols for naval tactical systems. The research requires a flexibly configured UYK-20 on the ARPANET. Specification and a functional design for the intercomputer protocol have been completed. We are awaiting the completion of the Dahlgren emulation facility interface to the ARPANET.

Since the completion of the basic PRIM system we have concentrated on making this technology available to military development projects expressing an interest and need. We have worked closely with the SDL at NOSC, with AFDSDC/LGSFE at Gunter AFS, and with the ARPA-sponsored National Software Works (NSW). For each of these development efforts PRIM has been able to provide

software tools compatible with the objectives of the user group. This effort entailed tuning the system, adding additional capabilities, and writing documentation to satisfy the stated user needs.

The success of PRIM as a development tool has generated sufficient interest that we are currently duplicating this environment with current off-the-shelf computer systems for installation in two military laboratories. The new PRIM is being built on a base of the DEC TOPS-20 timesharing System and the Nanodata QM-1 microprogrammable processor. The new systems are scheduled for installation at RADC and NOSC by October 1979.

QPRIM

A proposal to RADC consisting of a description of the newly proposed PRIM system has been written and accepted. The new system (QPRIM) utilizes recently purchased RADC TOPS-20 and QM-1 computer systems. In addition to duplicating the existing PRIM capabilities, QPRIM also allows a QM-1 stand-alone mode of operation. QPRIM provides RADC an enhanced emulation facility (and NSW tools) without compromising any of the stand-alone capabilities provided by Nanodata. The completion of QPRIM successfully concludes the remaining original goal of the PRIM project: to provide a reproducible system that allows easy transfer of this technology.

NOSC is participating in this development effort by providing ISI with their recently purchased QM-1. The NOSC QM-1 was installed at ISI in September 1978 and will be available for the completion of QPRIM development effort. This system will eventually be installed at NOSC, coupled to their existing TOPS-20 system, for a second copy of a QPRIM system.

The new components of the QPRIM system are the microvisor, driver and hardware interface. The microvisor implementation includes a modified micro-instruction set unique to QPRIM users. This new instruction set includes firmware paging logic compatible with TENEX and TOPS-20 paging mechanisms. The modified instruction set is compatible with user-level MULTI (Nanodata standard QM-1 instructions) and is being coordinated with the SMITE compiler activity at TRW (a higher level language for writing microcode for QM-1). The driver is being developed by the ISI system group for operation with TOPS-20. The PRIM system software is compatible with this new environment.

The implementation of QPRIM must be performed with no interference with TENEX users and a minimum of stand-alone TENEX time (the systems are currently scheduled for seven hours of preventive maintenance per week). Therefore

caution and exhaustive testing must prevail throughout the phases of development. The initial mode of operation is a stand-alone QM-1 remotely operated via TENEX. The memory interface will be tested with a spare DEC ME10 memory and the I/O bus interface will be tested with the aid of an existing I/O bus emulator. We can therefore develop and test the interfaces, the new Nanocode, microcode, and microvisor with TENEX user mode software.

The second phase is a stand-alone QPRIM environment which completes the checkout of the memory and I/O bus interface and microvisors. The QM-1 is directly coupled to the TENEX memory and I/O bus and is exercised with existing user mode software, with appropriate support from the TENEX monitor. A basic UYK-20 emulator is being developed for a test vehicle. The final phase utilizes TOPS-20 (ISIE) in place of TENEX system to continue to exercise QPRIM. A QM-1 driver, embedded in TOPS-20 monitor system, is the only new component. The QPRIM system will be duplicated and installed at NOSC and RADC.

The hardware interface is being implemented by Nanodata to ISI specification. The required hardware modifications reside entirely in the QM-1, and the hardware modules will be available as standard Nanodata options. The DEC components (external memory and I/O bus) are available as standard options for TOPS-20 systems (2040, 2050, and 1090T). Therefore future copies of QPRIM will not require the direct involvement of ISI; the interested user can purchase the entire system from DEC and Nanodata. ISI will continue to be responsible for the QPRIM software.

MULTIPLE MICROPROCESSOR EMULATION (MMPE)

There are a number of different schemes for interconnecting microprocessors--for example, communication over a shared bus, through common memory, or through pairwise connections. Each of these is aimed at achieving high throughput, high reliability, or both. We want to study which of these schemes works best for a range of applications. To do so, we will set up an emulation and measurement facility on the QM-1 to help carry out the evaluations.

The issues to be considered are the following:

1. What language should be used to describe the multimicroprocessor systems?

   For rapid experimentation and clear communication with others, the architectures being considered should be described in a high-level language. ISPS (under development at CMU) and SMITE (TRW) are the most advanced candidate languages, but neither is adequate to describe the interconnections

among the microprocessors. Extensions to one of the languages will be considered and recommendations made.

2. What should be measured?

   If the emulations are to provide useful feedback to the system architect, measurements relevant to the intended use of the machine need to be taken. A low-level facility will be included for measuring the number of times particular events take place and the length of various delays. Of greater interest will be a high-level facility for converting global questions about throughput into an efficient set of low-level measurements.

3. How should the code for the QM-1 be structured?

   We envision that all of the contexts for the microprocessors and the microcode that emulates the microprocessors will be kept resident in the control memeory of the QM-1. Relatively rapid switching, perhaps every (emulated) instruction cycle, will be possible. However, studies of bus loading and interference or studies of other queueing effects may dictate that the emulation be event-driven. It is also possible that a mixture of round robin and event-driven scheduling might be used for different parts of a complex emulation. We will have to determine what emulation strategies are required, and we will attempt to select the emulation strategy that best matches the interconnection scheme being studied.

RESULTS

- PRIM has successfully completed all its primary goals.
- The use of PRIM by SDL for Navy software development projects has steadily increased.
- At the request of RADC and NOSC ISI is duplicating the PRIM system (QPRIM) with currently available off-the-shelf computer systems.
- PRIM and UYK-20 are available as NSW tools.
- A paper, "PRIM System - A Framework for Emulation-Based Debugging Tools," was delivered at NCC 78.

FUTURE PLANS

The QPRIM system will be implemented and tested at ISI during FY79. QPRIM systems will be installed at RADC and NOSC in early FY80. Throughout this development we will continue to monitor development efforts with SMITE and ISPS, computer specification languages used for compilation of microcode, to maintain compatibility.

PRIM tools will continue to be made available to interested users with access to ISIC or NSW. A minimal effort will be expended for maintenance, user interface, and tuning of the PRIM exec. Additional emulation tools will have to be requested and funded by the interested user.

REFERENCES

1. *A Research Program in Computer Technology, Annual Technical Report,* ISI/SR-76-6, June 1976.

2. Gallenson, L. et al. *PRIM System: User Reference Manual,* ISI/TM-75-1.

3. Gallenson, L., A. Cooperband, and J. Goldberg, *PRIM System: Tool Builder's Manual,* ISI/TM-78-7.

4. Gallenson, L., A. Cooperband, and J. Goldberg, *PRIM System: UYK-20 User Guide,* ISI/TM-77-5.

5. Goldberg, J., A. Cooperband, and L. Gallenson, "The PRIM System: An Alternative Architecture for Emulator Development and Use," *Proceedings of the MICRO-10 Workshop,* October 5, 1977.

6. Goldberg, J., A. Cooperband and L. Gallenson, "PRIM System: A Framework for Emulation-based Debugging Tools," *Proceedings of the AFIPS National Computer Conference,* Vol. 47, 1978.

8. PROTECTION ANALYSIS

Research Staff:
Jim Carlstedt
Richard L. Bisbey II
Dennis Hollingworth

Support Staff:
Debe Hays

The Protection Analysis Project was initiated at ISI in September 1973 to further the understanding of operating system security vulnerabilities and, where possible, to identify automatable techniques for detecting such vulnerabilities in existing operating system software. The primary goal of the project was to make protection evaluation both more effective and more economical by decomposing it into more manageable and methodical subtasks, drastically reducing the requirement for protection expertise and making it as independent as possible of the skills and motivation of individuals relative to security expense. The project focused on near-term solutions to the problem in an attempt to have some impact on the security of systems that would be in use over the next ten years.

A general strategy was identified, referred to as "pattern-directed protection evaluation" [Carlstedt 75] and tailored to the problem of evaluating existing commercial operating systems. The approach provided a basis for categorizing protection errors according to their security-relevant properties; one such category was successfully applied to the MULTICS operating system, resulting in the detection of previously unknown security vulnerabilities [Bisbey 78]. Several ISI reports describe the error categories and their corresponding detection techniques [Bisbey 75], [Bisbey 76], [Carlstedt 76], [Hollingworth 76], [Carlstedt 78a].

STATUS

Between June 1977 and September 1977, the report describing "serialization" errors was completed [Carlstedt 78a]. An annotated bibliography and index was also prepared [Carlstedt 78b] along with a report summarizing the work [Bisbey 78].

In September 1977, the project was terminated by mutual agreement of ARPA IPTO and ISI. It was concluded that the pattern-directed error detection approach to security evaluation had been shown feasible, and lacking active interest by computer vendors, continued research by ARPA was not warranted.

REFERENCES

BISBEY 75 Bisbey, Richard, II, G. Popek, and J. Carlstedt, *Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time,* USC/Information Sciences Institute, ISI/SR-75-4, December 1975.

BISBEY 76 Bisbey, Richard, II et al., *Data Dependency Analysis,* USC/Information Sciences Institute, ISI/RR-76-45, February 1976.

BISBEY 78 Bisbey, Richard, II and D. Hollingworth, *Protection Errors in Operating Systems: Final Report,* USC/Information Sciences Institute, ISI/SR-78-13, May 1978.

CARLSTEDT 75 Carlstedt, J. et al., *Pattern Directed Protection Evaluation,* USC/Information Sciences Institute, ISI/RR-75-31, June 1975.

CARLSTEDT 76 Carlstedt, J., *Protection Errors in Operating Systems: Validation of Critical Conditions,* USC/Information Sciences Institute, ISI/SR-76-5, May 1976.

CARLSTEDT 78a Carlstedt, J., *Protection Errors in Operating Systems: Serialization,* USC/Information Sciences Institute, ISI/SR-78-9, April 1978.

CARLSTEDT 78b Carlstedt, J., *Protection Errors in Operating Systems: A Selected Annotated Bibliography and Index to Terminology,* USC/Information Sciences Institute, ISI/SR-78-10, January 1978.

HOLLINGWORTH 76 Hollingworth, D. and Richard Bisbey II, *Protection Errors in Operating Systems: Allocation/Deallocation Residuals,* USC/Information Sciences Institute, ISI/SR-76-7, June 1976.

9. DISTRIBUTED SENSOR NETWORKS

Research Staff:
Danny Cohen
Yechiam Yemini
Jeffrey Barnett

Support Staff:
Debe Hays

1. INTRODUCTION

1.1 Background

In the mid-1980s military effectiveness will depend upon rapid collection, interpretation, evaluation, and dissemination of information. By that time a major portion of all strategic and tactical information will be computer-processable, and automated information processing will replace traditional manual methods. Achieving an integrated information system encompassing the full spectrum of command, control, and communications (C^3) functions will require a major architecture development effort in order to integrate the corresponding C^3 technologies. The Distributed Sensor Networks (DSN) research will contribute to the development of integrated C^3 information systems in two forms: first, smart sensor networks will form an essential part of future integrated military information systems, and second, DSNs offer a microcosm of the future operation of integrated C^3 systems.

The immediate objective of the DSN project at ISI is to develop an architectural framework for constructing DSNs. An immediate application would be to design DSNs for low-flying aircraft and cruise-missile detection and tracking. The framework that we develop should also serve as a model for other DSNs (e.g., DSNs to track submarines or to identify hostile artillery sources).

DSN research is directed towards both the individual development of the three technologies (i.e., sensors, communication, and information-processing) and their integration. That is, besides problems of developing better sensors, designing computer communication protocols to support distributed cooperative processing, or developing intelligent signal-processing systems using Artificial Intelligence technology, problems exist at the interface between the different technologies. For instance, high-level intelligent processes should be able to influence the decisions of the lower-level communication mechanisms to permit adaptive network policies.

The major problem as far as sensor technology is concerned is to develop sensor capabilities such as those needed to detect low-flying aircrafts. Research is being conducted to develop acoustic-arrays, low cost radars, and infrared sensors to identify low-flying targets. The limitations upon the range, capabilities, and accuracy of such sensors require that a DSN should deploy a multitude of low cost sensors to provide good coverage.

Deploying a large number of sensors whose individual observations can provide only partial information, blurred by noise, requires a satisfactory communication mechanism between sensors to support collection and integration of the sensors' data. The development of Packet Radio Networks (PRNETs) will provide a suitable communication technology. Once equipped with a Packet Radio Unit (PRU) each sensor could exchange information with other network members. Once information is collected and communicated it must be integrated and evaluated. DSNs will have to support sophisticated, intelligent, distributed processing capabilities to provide those functions.

The above description, crude as it is, is sufficient to demonstrate that the development of DSNs should produce solutions to major problems to be faced by designers of future C^3 systems. It is in this sense that the DSN research transcends its immediate target of providing a detection and tracking system for low-flying targets.

1.2 Goals

At this initial stage of research, the main goal of the DSN project is to develop a conceptual model for structuring DSNs. How should sensors' signals be transformed all the way into high-level intelligence in real time? How should the different processes involved cooperate to optimize the performance? What are the important performance measures? What are the important design parameters? What are the significant tradeoffs? These are some of the major problems requiring solutions. The attempt to develop an overall model must be supported by a concurrent effort to isolate important problems that may be treated separately. That is, a top-down approach should be supported by a bottom-up solution of significant smaller problems. We have tried to combine the two approaches in choosing the directions for our research.

1.3 Problems attacked

Our research has been directed in part to develop a conceptual model of DSNs. The questions that we have tried to answer have been: How should DSNs be configured? What are the functions to be supported by DSNs? How should those functions be modeled? What are the important issues to be resolved? How should DSNs be described?

The major problem as far as sensor technology is concerned is to develop sensor capabilities such as those needed to detect low-flying aircrafts. Research is being conducted to develop acoustic-arrays, low cost radars, and infrared sensors to identify low-flying targets. The limitations upon the range, capabilities, and accuracy of such sensors require that a DSN should deploy a multitude of low cost sensors to provide good coverage.

Deploying a large number of sensors whose individual observations can provide only partial information, blurred by noise, requires a satisfactory communication mechanism between sensors to support collection and integration of the sensors' data. The development of Packet Radio Networks (PRNETs) will provide a suitable communication technology. Once equipped with a Packet Radio Unit (PRU) each sensor could exchange information with other network members. Once information is collected and communicated it must be integrated and evaluated. DSNs will have to support sophisticated, intelligent, distributed processing capabilities to provide those functions.

The above description, crude as it is, is sufficient to demonstrate that the development of DSNs should produce solutions to major problems to be faced by designers of future C^3 systems. It is in this sense that the DSN research transcends its immediate target of providing a detection and tracking system for low-flying targets.

1.2 Goals

At this initial stage of research, the main goal of the DSN project is to develop a conceptual model for structuring DSNs. How should sensors' signals be transformed all the way into high-level intelligence in real time? How should the different processes involved cooperate to optimize the performance? What are the important performance measures? What are the important design parameters? What are the significant tradeoffs? These are some of the major problems requiring solutions. The attempt to develop an overall model must be supported by a concurrent effort to isolate important problems that may be treated separately. That is, a top-down approach should be supported by a bottom-up solution of significant smaller problems. We have tried to combine the two approaches in choosing the directions for our research.

1.3 Problems attacked

Our research has been directed in part to develop a conceptual model of DSNs. The questions that we have tried to answer have been: How should DSNs be configured? What are the functions to be supported by DSNs? How should those functions be modeled? What are the important issues to be resolved? How should DSNs be described?

Another direction of our effort to develop a general DSN model has been to consider the problem of performance objectives. Here we have asked the following questions: What are the important performance parameters? How should we evaluate and compare the performance of alternative DSN designs under different target configurations?

Concurrently with our research into problems of modeling, we have developed a software testbed to establish a concrete simulation model of DSNs.

In addition to problems of modelling we have attacked the positioning problem. PRNet units may use a simple time-stamping protocol to measure their relative distances. These measurements, which are virtually free, may be used to compute the actual location of the participating units. The positioning problem is how to locate the position of plane objects from sparse inaccurate measurements of distances between some pairs of objects. In addition to the development of a solution to this important problem, the object of this research has been to develop a concrete understanding of issues involved in decentralized processing of algorithms.

1.4 Results obtained

1. We have developed a methodology to model DSNs. We have isolated and defined the boundaries of some major design problems.

2. We have developed a mathematical taxonomy of DSN performance parameters and a methodology to establish performance evaluation index.

3. We have developed a simulation model, the software testbed, within which different concepts and alternative DSNs designs may be tested and developed.

4. Finally, we have developed new mathematical theories to solve the positioning problem, as well as algorithms to locate the positions of objects in the plane from distance measurements and to test the solution for error sensitivity. We have implemented sophisticated software to implement those algorithms.

2. DEVELOPMENT OF A DSN MODEL

A major objective of the DSN project is the development and evaluation of models of sensor-based systems. Our approach is to analyze the models when possible and to augment the analysis with simulation when necessary. The expected results of this process are:

1. Conclusions about behavior characteristics of DSN systems as a function of their architecture.

2. Comparisons of behavior characteristics as functions of choices of architecture.

3. Design of protocol schemes for intercomponent communications.

In order to properly compare and evaluate systems, it is necessary to consider their intended use and the environment in which they operate. For example, a system that performs extremely well only near critical regions is probably more valuable than a system with mediocre but uniform performance. Also, environmental effects can alter the performance of the systems in substantive ways, e.g., wind may influence the performance of acoustic sensors.

In order to consider the above kinds of effects properly, we have generalized the DSN model to include both components that evaluate sensor and tracking data (perhaps generating responses) and the environment as well. The additional components could be command and control systems, control mechanisms that focus illuminators and detectors to specific areas, and guidance systems that use the information gathered by the rest of the system. Thus the generalized problem statement not only has sufficient breadth to make system comparison possible, but also is rich enough to cover the intrasystem feedback that will surely exist in future DSNs. Even in primitive systems being considered today, feedback is an important issue--given limited bandwidth communications, it is necessary to transmit only the most essential data, and in a decentralized system this can be determined only cooperatively, because no single site has available the necessary global viewpoint.

We are currently developing a language with which to describe DSN models; this language should serve two purposes:

1. Allow analysis of DSN models.

2. Be executable by a simulation system.

In order to serve both purposes, the language includes an assertion sublanguage as well as features to express procedures. The former allows the modellers, using mathematical notation, to describe abstract properties of system components. Another of its important uses is the description of the DSN *framework*, an abstract model that covers any DSN system, including minimal descriptions (necessary conditions) for each category of component or object that can appear in a DSN system. A model for a particular system is formed by adding specific assertions

and procedures to the framework to complete the model. Since the language is based upon an abstract data type paradigm with hierarchical decomposition, this works quite nicely--for a particular model, embellish the framework (add lower levels to the hierarchy) with system-specific information; for other details, use the minimal descriptions provided by the framework. The remainder of this section describes the major problems facing the modelling activity, the modelling language, methodology to compare systems, and directions of future work in DSN modelling.

2.1 The Problems

Perhaps the most important problem facing the DSN modeller is developing a formal framework in which to embed the models. The framework must provide a definition of the DSN space, from which are drawn problems (scenarios) that systems must confront as well as possible architectures and configurations that they may employ. Further, the framework must provide one or more objective functions to evaluate system performance and make comparison among systems possible. More than any other activity, consideration of an appropriate objective function brings issues into sharp focus because it demands answers to the two questions: What is to be measured? and What is important? Progress on this task is reported in Section 2.3 below.

Many important questions about sensor systems exist even in nondecentralized configurations. Below is a partial list of these questions, the answers to which must be reflected in the DSN framework and models:

1. What are the models of target behavior? What are the constraints on target motion? What are the correlations between a target's behavior and its mission?

2. What trails do targets leave? What are the physical effects caused by targets? How do these effects relate to the target's behavior?

3. What trails do sensors see? To what physical phenomena does a sensor react? What is the correlation?

4. What data do sensors output? What is the (statistical) relation of sensor output to target behavior?

5. What are the speed/quality/cost tradeoffs? Within a particular sensor technology? Among different technologies?

6. What is the state of the art in tracking multiple targets?

7. What is the state of the art in identifying targets and evaluating their threat?

8. How is the sensor output to be used? By whom?

9. How is higher level knowledge (e.g., intelligence) used?

10. What are the 5- to 15-year future answers to the above questions?

Questions 8 and 9 are the most interesting as well as the hardest to answer, for several reasons. First, the prospects of decentralization, cheaper hardware, and progress in knowledge-based systems technology are changing the answers. Second, the sensor technology community has traditionally deferred these two questions, particularly 9, and defined their objectives in terms of tracking quality for a given target model. Further, questions 6, 7, 8, and 9 together cover important sensor system issues that are only now being fully addressed: How, in an automated or semiautomated manner, is sensor information to be evaluated and interpreted and what is the possible range of system-generated response?

The prospect of decentralization may have a major impact on the possibilities here. The first reason is obvious; in a decentralized system, information about a target's existence and behavior must be shared. Therefore, the decentralized system has an opportunity to observe the target over longer time spans and hence to observe more behavior on which to base an evaluation. The second reason, more speculative, relies on the belief that, to work at all, decentralized systems must employ more higher level knowledge than conventional systems. If the problem of knowledge incorporation can be understood well enough to perform traditional functions, then progress will have also been made on knowledge incorporation technologies for handling the newer problems.

Several other problems and issues are raised by the prospect of decentralization.

1. At what levels of functionality can sensor systems be decentralized (e.g., illumination, detection, tracking, etc.)? What criteria should be used?

2. What kind of communication scheme is most appropriate? Are application-specific protocols necessary?

3. Are special software tools necessary to implement DSNs? What form do these tools take (e.g., special-purpose programming languages, applications packages, etc.)?

4. How are vulnerability and reliability enhanced? What are the tradeoffs?

5. What "nontraditional" components will the network incorporate (e.g., command and control systems, observers with portable terminals, etc.)?

6. How should algorithims be decentralized? Some of the issues are time delays and partial information.

7. How will the user employ the system? That is, what is the interaction paradigm if the user needs to understand output, including verification, or the user needs to direct or focus the activity of multiple components?

8. What are the bandwidth/performance/cost tradeoffs?

9. What advances are expected in communications technology over the 5- to 15-year time frame?

While issue 3 may at first seem trivial, it raises a very serious problem. Protocol schemes in current vogue have been structured so that the internal workings of lower levels are invisible to higher levels. Assuming component and communication technologies do not overwhelm the problem with unexpected speed increases, this will not do. It is crucial that applications-level decisions by knowledgeable components affect at least priorities and flow control in the network. If this is not the case, a DSN fails quickly on overload. A solution to this problem involves developing a methodology for intimate interaction between application-level programs and network software, traditionally part of the protected operating system. A methodology that does this gracefully and still allows reasonably structured software development is one of the major contributions that computer science can make to the sensor technology community. In particular, this is a central issue to be addressed in constructing the DSN framework and system models.

2.2 A Methodology to Describe Systems

The DSN project is currently developing a formal language to encode the DSN framework and models of DSN systems. The language borrows heavily from the abstract data type paradigm; that is, all knowledge about a particular class of objects is encapsulated (centralized). The knowledge may include structural specification (such as data type declarations), operators that manipulate objects of that class, and assertions that describe general constraints.

Further, the language permits hierarchial decomposition of the object classes. A parent class provides default information (data and procedures) about all objects

in its class. A subordinate class provides more detailed information about objects in the subclass. There is a natural consistency requirement that the objects in the subclass satisfy all assertions about objects in the parent class.

So far nothing differs from current practice with abstract data type languages such as SIMULA or CLU. However, DSN pushes hard on an old problem: it is often necessary and/or desirable for a class to have more than one parent class. Consider, for example, a natural decomposition that includes the classes CRAFT (aircraft, ships, etc.) and EQUIPMENT (illuminators, defractors, etc.) What then is a "cruise missile"? Surely it is a CRAFT and may even be tracked by the rest of the system, and just as surely it can act as part of the sensor net in detecting other craft. Also, consider the EQUIPMENT class. It seems natural to divide it into subclasses based upon function--this is even necessary if the problem of where and how to decentralize is to be explored. But with this natural division, a radar must have multiple parent classes of at least ILLLUMINATOR and DETECTOR and--if it is sophisticated--TRACKER. The problems with multiple parent classes are name conflicts (a syntactic problem) and functional ambiguity (a semantic problem). The functional ambiguity arises when an object is treated as if it can be made to function as a simple object, i.e., an object with a single parent. For example, if a radar is focused as an illuminator only object to a particular sector, it loses its detection capability in all other sectors. This is just the hidden side effect issue brought to the applications level.

Given the number of object classes in the DSN environment, it is not feasible to explicitly include all possible combinations of the classes in the framework--the number of combinations quickly gets out of hand. Nor does it make sense not to use some sort of hierarchial decomposition scheme--the jumble of details becomes unmanageable.

A preliminary framework for DSN has been drafted and classes with potential multiple parentage have been identified. Three cases have arisen: (1) people--intellegence agents and users--perform chores similar to that performed by equipment, (2) some equipment, e.g., a tracking radar, performs the aggregate tasks of several smaller pieces of equipment, (3) craft, e.g., a cruise missile, can perform as craft and sensor equipment. To lessen problem 1 some human functions are modeled as if they are equipment functions. Problem 2 is being approached either by suppressing certain functions (e.g., assuming that a radar's illuminator function can be safely ignored by the rest of the model) or by modelling the separate functions as separate pieces of equipment connected by a tightly coupled network. A solution to problem 3 remains elusive. For the present, the language provides the ability for specifying multiple parent classes i.e., class intersection. For this particular case, there is no serious problem; however, as a general language feature, a potential exists for undesirable effects to occur.

In order to allow the language to serve its intended functions--framework and model descriptions--an assertion sublanguage is provided. Often, the framework and higher level object classes in a model need not provide detailed procedural knowledge. Instead, assertions written in an extended first-order predicate calculus describe the effects and constraints on objects and associated operators. The assertion sublanguage has the advantage that it lends itself directly to analysis. Further, it makes sense to say that a child class must be consistent with the parent's assertion while the alternative, a requirement of procedure-to-procedure consistency, would allow no room to supply missing details.

In addition to serving as a tool for modelling and description, the language must also be used for simulation. The two most important decisions in selecting a simulation regime are how to handle time and how to communicate information among the modelled objects. For DSN, discrete time simulation is the most natural because knowledge about craft and equipment is generally given as difference equations or as state machines. Also, discrete time simulations (particularly event-driven) are in general the most efficient.

The information communication regime selected for simulating DSN is based upon the Hearsay II blackboard scheme developed at CMU. The blackboard is a global data structure used to communicate information among parallel cooperative processes, which use a pattern to detect interesting data on the blackboard. If the pattern is matched, the processes are scheduled for execution with the blackboard data as arguments. Thus, process activation can be called data-directed or "demon" invocation.

For purposes of DSN, each process is associated with an object, i.e., a craft or piece of equipment. Because of the decentralization, it is obvious that each process should be able to see only the data that is appropriate--not the entire blackboard--or the simulation would be centralized. Thus, the pattern for each process restricts the data seen by the process to its actual focus, comprising location and physical characteristics. For example, a radar can detect objects only within a given radius and is insensitive to acoustic communications. Thus, even though a microphone sensor may be sensitive to sounds from behind a physical barrier, a radar will not. The pattern also expresses "wake up" conditions. A wake up condition (or interrupt) is an external event or situation that can alter the normal flow in a process.

There are many advantages to using a blackboard scheme for DSN simulation, chief of which is decoupling complexity: that is, a process can receive its arguments and produce its results with no knowledge of what other process(es) will use its output. For example, a process that generates a trajectory for an

airplane need not know (or care) whether the sensing device in this simulation is a microphone, a radar, or both. Another advantage of a blackboard and data-driven activation is the enhanced ability to provide a good trace package. The trace and debugging routines can use the pattern activation scheme to find interesting data for display. It is necessary for a good simulation system to provide not only summaries, but also a methodology for examining its step-by-step inner workings.

The blackboard is also used to represent network states, which means that characteristics of a network behavior can be modified by changing the process that simulates it without modifying the nodes on that network. Thus, many experiments about system performance can be performed with minimal changes to the system model.

The framework and modelling language allow for two types of entities: objects and artifacts. The former are such things as equipment, craft, and geographical targets. The latter are time-varying entities such as virtual objects and states of objects. A virtual object is an entity that exists for convenience of description; for example, a set of equipment not a priori aggregated may dynamically take up a common goal. Such an aggregation would be called a cluster. Some of the members of the cluster may be assigned specific tasks such as intercluster communication or tracking. For the lifetime of the cluster, it would be treated as an object and its state maintained by the system.

It is the artifacts that are kept on the simulator's blackboard. Each artifact is comprised of several attribute/value pairs. At a minimum, the artifact must identify its type, e.g., craft state, and its period of validity. Other attributes are specific to the type. The language provides a mechanism for describing the constituent attributes for each type, including data type restrictions for the attributes as well as the ability to attach special procedures to instances of the type. The procedures can provide computations depending upon details of data storage that are not of concern to users of the type. Artifact type declarations can be hierarchically decomposed; for example, the subtype acoustic sample adds the attributes frequency and loudness to those specified for the parent type, sample.

The language provides for the definitions of operators. An operator is a process scheme. The definition describes under what circumstances to create processes; for example, an operator associated with some class of airplane specifies an instance for each particular airplane of that class that is to participate in the simulation. An operator definition provides in addition an activation pattern for the processes and computation component.

In general, an operator interrogates one or more knowledge bases to obtain data and algorithms specific to its associated object. Knowledge bases are another

example of an abstract data type mechanism in the language; they are used as structured constants that contain the model's static knowledge about object classes and specific objects. In a sense, an operator is an interpreter and the knowledge bases are the program text.

Thus, the language provides two kinds of hierarchically decomposable abstract data types--artifacts and knowledge bases. The former are instances of time-varying states and objects and the latter instances of static information. Artifacts appear on the blackboard, are "seen" by operator patterns, and are created and manipulated by operators; knowledge base instances are not. Also note that the operators are not associated as functions of a particular data type; they may access knowledge bases and artifacts across many levels and branches of the hierarchies as need be. For example, an operator generating a craft trajectory needs to view and/or manipulate artifacts from many types including mission, craft state and environment, and it must access the knowledge bases describing craft characteristics and target location information.

The language provides many features for the more program-specific chores of generating the framework and models. Several built-in types, with associated operators, are included--integer, real, and complex are the built-in numerical types and boolean and identifier are the built-in nonnumerical types. Other types can be constructed from the built-in types and other constructed types, e.g., lists, directed products, and structures. Also, a primitive type definition facility is provided for more complex type declarations when operators need to be associated with instances of the type. An example of such a primitive type is coordinate location, whose associated operators are distance and heading between two locations. The primitive type facility allows the declaration of physical dimensions, such as mass and velocity, and the declaration of units within the dimensions, such as kilometers and miles. This sort of facility is very important in simulation and modelling languages because experience has shown that when several people participate in model construction a large number of errors occur without it. It is the simulator's responsibility to assure that expressions are dimensional-compatible and to automatically perform the necessary units conversions.

The DSN project is also investigating the construction of a translator whose job would be to perform consistency checks on the framework and models and translate the models into an executable form. The simulator will operate either in SIMULA or INTERLISP; a firm decision has not yet been made. The former offers the advantages of efficiency and compatibility with the abstract data typing, while the latter's advantage is flexibility. One major consideration is the creation of a display package for presentation of simulation results; the simulation system will be a a software testbed for testing and evolving possible DSN architectures, and as

such it must have the wherewithal to present usable results to many who are not directly a part of the computer science community. Section 4 describes our currently available testbed.

2.3 Methodology to Assign Performance Measures

The purpose of this research has been to develop a method by which the performance of DSN can be paramaterized. The methodology first provides a means of comparing the performance of alternative designs of DSN, under various target configurations. By quantifying the performance, it is also possible to examine simulation and analytic models of DSN and identify critical design parameters and tradeoffs.

We have considered first the problem of single-target tracking. We have classified the different types of "tracking errors" and the different methods to evaluate those errors. This study differs from the classical development of objective functions mainly in its scope and generality. Classically, an objective function of an estimation algorithm is based on a certain target model that underlies the whole scheme. The problem is then to chose the best scheme *with respect to the given model*. Since in a DSN the choice of target model is but another design problem, we must develop a performance evaluation scheme independent of the target model of the tracking algorithms. It is primarily in this sense that our scheme differs from the classical approach.

Once we have established a scheme to evaluate single-target tracking errors, we proceed to the multi-target scenario. Here new problems arise when we consider the problem of associating estimated tracks with targets. For instance, how should we evaluate the performance of a DSN that cannot resolve two targets unless they are at least a mile away, or going in different directions? How should we evaluate the performance of a DSN that produces many false tracks when confronted with dense clusters of targets? These and other problems of multi-target tracking evaluation have been successfully resolved by using techniques of clustering theory.

Now that a performance evaluation methodology has been established, it will be possible to examine analytic and simulation models of the major design parameters and tradeoffs.

2.4 Future Research

Several problems occupy the immediate future of the DSN modelling effort, the two most pressing of which are completing the language design and encoding the framework in that language. Completing the framework description requires

thoroughly understanding the present and future possibilities for sensor equipment and higher level components, developing an outline of communications requirements, and choosing objective functions.

Given progress on the above items, the next steps are implementing the new software testbed and developing communication schemes for the systems. The most important and intellectually challenging problem at this stage is the investigation of methodology to incorporate application-level knowledge into communications protocols.

Future work will consist of using the software testbed and analysis techniques to evaluate possible DSN architectures, the result of which will be a categorization of the available options and the tradeoffs between the options and system performance and costs.

3. POSITION LOCATION

3.1 Definition of the problems

The positioning problem arises when it is necessary to locate a set of geographically distributed objects using measurements of the distances between some object pairs. In a PRNet, for instance, any two network members that can talk to each other may use a simple time-stamping mechanism to measure the distance between them; a distance measurement protocol may then be developed. The problem is whether and how the distance measurements can be used to determine the geographical location with respect to a given system of coordinates.

A knowledge of the precise location of each network node is crucial to the operation of DSNs. The data collected and interpreted by different sensors may be correlated and integrated only if we know their precise location. A position-locating system may be invaluable to the operation of a fleet of vehicles, each equipped with a Packet Radio Unit. For example, monitoring the location of a fleet of security vehicles, aircraft, a tank division, or a flock of missiles could all be assisted by a position-locating system. Clearly a positioning system would be an important service to PRNet users.

A few problems must be solved before a good positioning system may be developed:

1. Efficient algorithms should be developed to determine the location of objects by means of distance measurements.

2. Conditions should be established under which there exists a unique solution.

3. Conditions should be identified under which there exists a finite number of solutions. It should also be understood how to transform one solution into another.

4. Conditions should be established under which the solution is insensitive to small measurement errors.

5. Tight bounds upon the accuracy of the solution should be determined.

6. Redundant measurements should be used to eliminate errors.

However, while the formulation of the problems is simple, the mathematical and algorithmic intricacies of deriving solutions are perplexing.

3.2 An approach towards a solution

There are two possible approaches to the positioning problem. The "brute force" approach is to describe the constraints imposed by the distance measurements in terms of a system of equations in the unknown location coordinates of the objects. A large system of sparse nonlinear equations is obtained. It is possible to apply iterative methods that may converge to a possible solution if we start with a good initial guess. This approach leads to cumbersome computational processes that may not be applicable to large systems, may never converge to a solution, may converge to a wrong solution, cannot be properly decentralized, and finally cannot incorporate other information (which may be provided by other sources such as human operators) to alleviate the computational task.

The other alternative is to develop an intelligent method of solution that identifies proper solution strategies from the structural combinatorial properties of the problem. The idea is to develop an incremental algorithm that identifies and locates the position of subsets of objects incrementally until a full solution is obtained. This is but another application of the age-old "divide-and-conquer" rule. Since the processes of identifying subsets of objects whose relative positions can be solved may occur simultaneously at different parts of the network, this approach leads to an algorithm that may be implemented in terms of decentralized computations. In addition, an incremental method should be intelligent enough to identify ill-conditioned problems, information insufficient for generating a solution (and what additional measurements should be performed). It is also possible to identify and generate all possible solutions when the set of solutions is sufficiently small, letting a human operator select the most reasonable solution or

guide the system towards the most reasonable alternative. In short, the approach through incremental algorithms offers a range of attractive features.

The major difficulty with the incremental approach is that before an intelligent algorithm may be constructed, the required intelligence must be developed. For instance, how should the algorithm identify subsets of objects that can be located? How should it locate those subsets once identified? These and other major problems must be properly addressed. The difficulty is far from being trivial, for the positioning problem has been attacked (though in other forms) by many prominent mathematicians and scientists over a period of a few centuries, yet most problems that have been listed in the previous section are far from having a satisfactory solution.

We have carefully studied the results obtained by generations of researchers, added our own novel contributions, and incorporated this knowledge into a sophisticated position location algorithm possessing most of the desirable properties expected from the incremental approach. In what follows we shall describe these results.

3.3 Description of results

Our major thrust has been to develop a mathematical theory to understand the positioning problem and then apply the theoretical study to develop intelligent algorithms. Accordingly, our results fall in two categories--theoretical and algorithmic.

3.3.1 *Theoretical results*

The positioning problem may be modeled as follows. Objects are represented as vertices of a graph; two vertices are connected if the respective objects have measured the distance between them. The measurement process may be thought of as an assignment of lengths to the edge of the system graph. A graph together with an assignment of edge-lengths is called a *structure*. If we identify edges with rigid bars and vertices with pin-joints, then our "structures" correspond to pin-jointed mechanical structures. The problems of positioning are analogous to problems of constructing skeletal structures. For instance, a positioning problem that is not well determined (i.e., possesses a continuum of solutions) corresponds to a nonrigid mechanical structure; a positioning problem whose solution is sensitive to measurement errors corresponds to a mechanical structure that is not infinitesimally rigid (i.e., admits nontrivial infinitesimal perturbations), and so on.

In view of the analogy above, we have adopted the terminology and some major results of the theory of structural rigidity. Unfortunately, the main concern

of structural engineers has been to evaluate the strength of structures that have already been constructed. Very little has been done to attack the problem of how to construct structures, which is our main interest.

The major results of rigidity theory are a few equivalent characterizations of rigidity and a recent theorem which establishes that rigidity (a geometric property) is almost always identical to a combinatorial property of having enough well distributed equations (measurements) to find the unknown coordinates. This last combinatorial property, which we call *stiffness*, is independent of the actual distances measured; it is a property of the system graph only. The rigidity theorem is invaluable for the study of incremental positioning algorithms, since it provides us with simple criteria to recognize systems whose positions may be located.

Our novel theoretical results include the following:

1. A new criterion for rigidity of positioning systems, which may be implemented in terms of a fast algorithm (thus effectively determining whether or not a given system is error-sensitive or not).

2. A set of combinatorial theorems relating stiffness to other graph properties. For instance, we developed methods to tear graphs into stiff components so that positioning problems may be divided before they are conquered. Other significant heuristics were developed and many intuitively appealing heuristics were shown to be false through difficult counterexamples.

3. A fast algorithm to determine whether a given graph is stiff (i.e., represents a constructible positioning system).

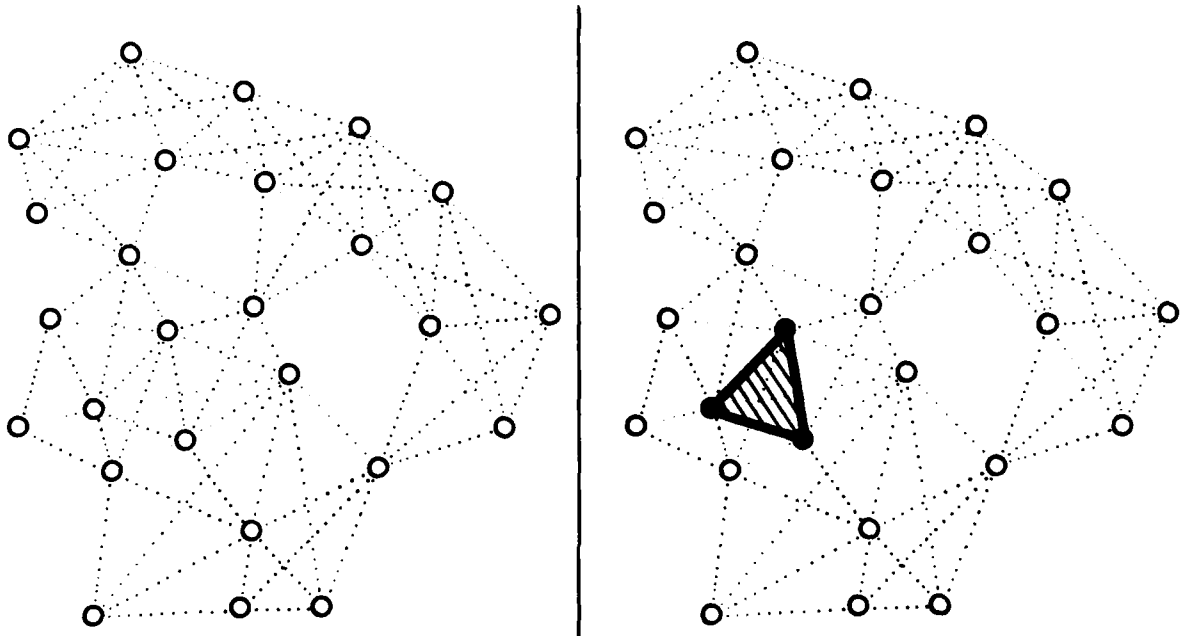
4. A novel theory of stiff-hypergraphs within which it is possible to formulate develop and prove properties of any incremental position location algorithm.

5. In addition, as a byproduct, we have derived some results that have little pragmatic value but bear significant theoretical implications. We have shown some problems of positioning are NP complete (that is, as difficult as a whole class of classical problems for which no polynomial time algorithms are known). For instance, the problem of finding the correct solution of a triangulated positioning system (i.e., one constructible from triangles), among all possible relative reflections of the triangles with respect to each other, is NP complete. This implies in particular that the problem of determining whether a solution to a positioning problem exists is NP complete.

3.3.2. *Algorithmic results*

The heuristics developed through the theoretical study of positioning problems were implemented into sophisticated software designed to support incremental positioning algorithms. The idea behind our position-location algorithms is simple. We conceive of an incremental position-location algorithm as a mechanism that identifies stiff substructures of a structure and *welds* these substructures together. The welding process corresponds to a solution of the relative location of points belonging to a substructure, relative to a local coordinate system. The welding process is applied recursively to the structure until all parts of the structure are welded together (i.e., all locations are known). This incremental welding process is demonstrated in Fig. 9.1.

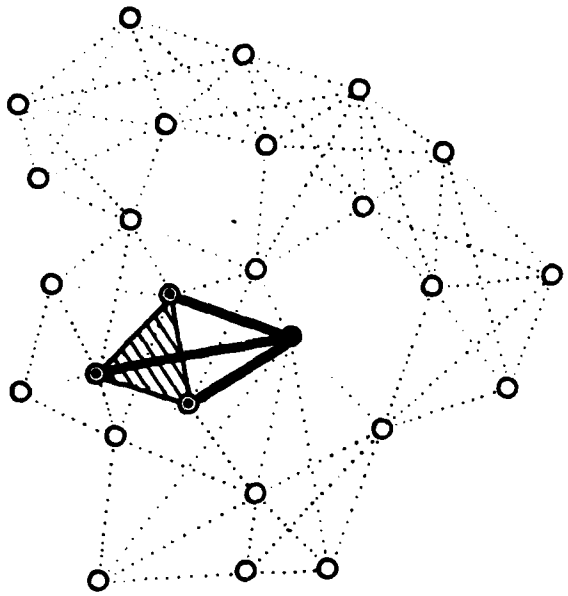
We have implemented the concepts and tools required to develop positioning algorithms within a SIMULA system. A few welding operators have been implemented and tested successfully on some positioning problems.



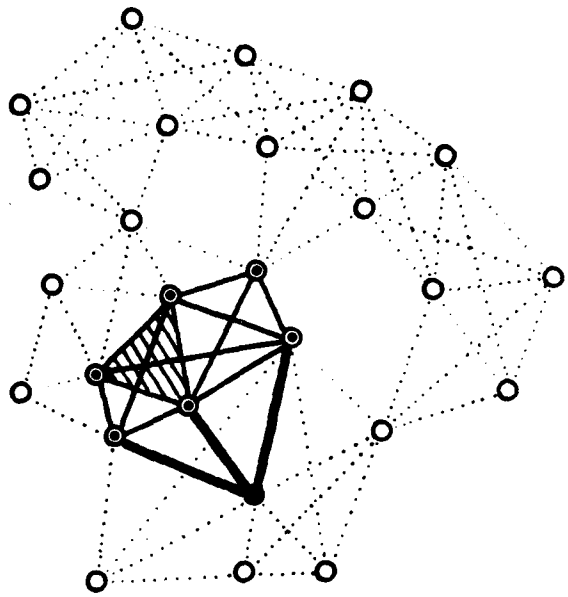
(a) A position-location problem.

(b) A base triangle is identified.

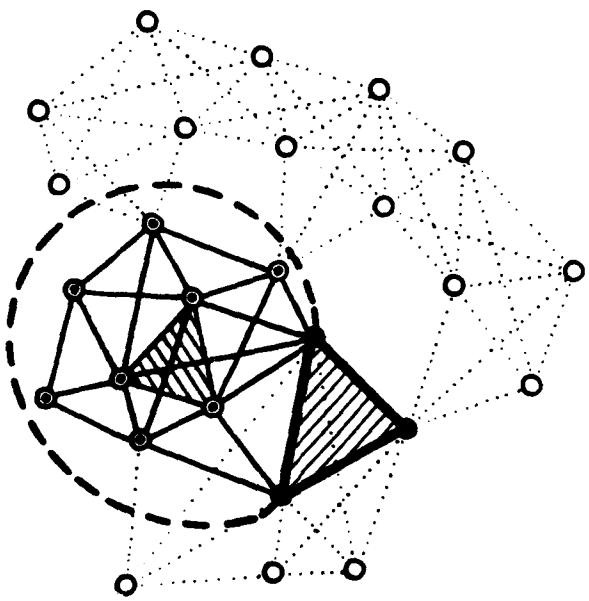
Figure 9.1 Incremental "welding" algorithms



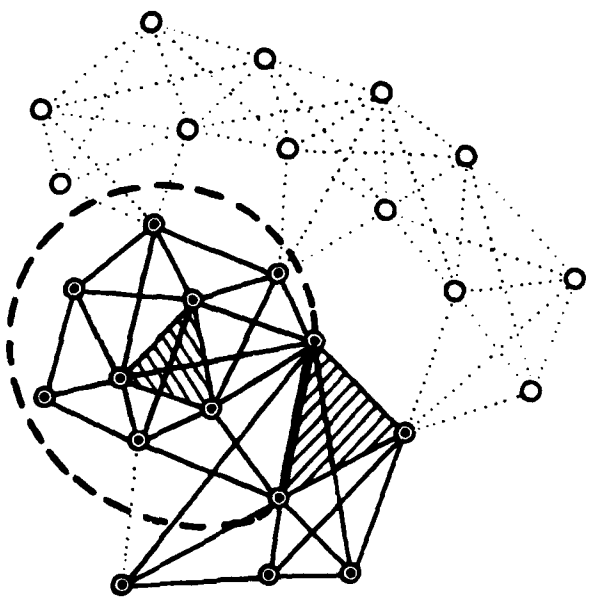
(c) A 3-point (one connected with 3 edges to points already positioned) is "welded" to the triangle.



(d) More 3-points are welded into the body until no more 3-points are available.

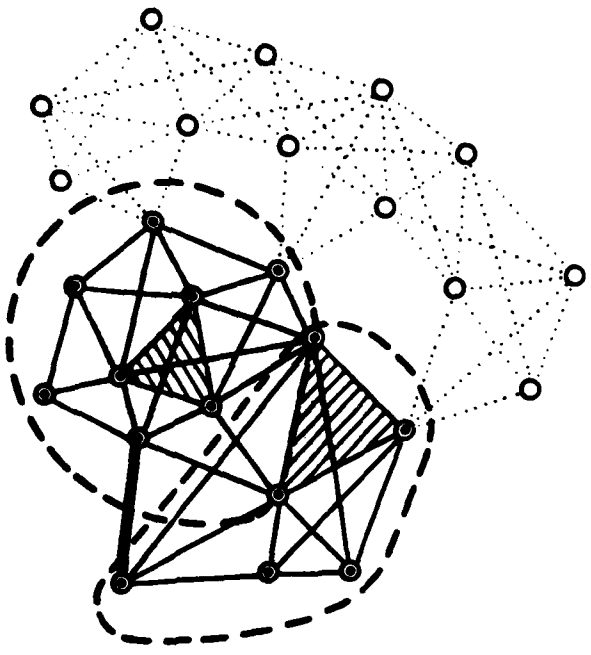


(e) A new base triangle is identified and a new body is established.

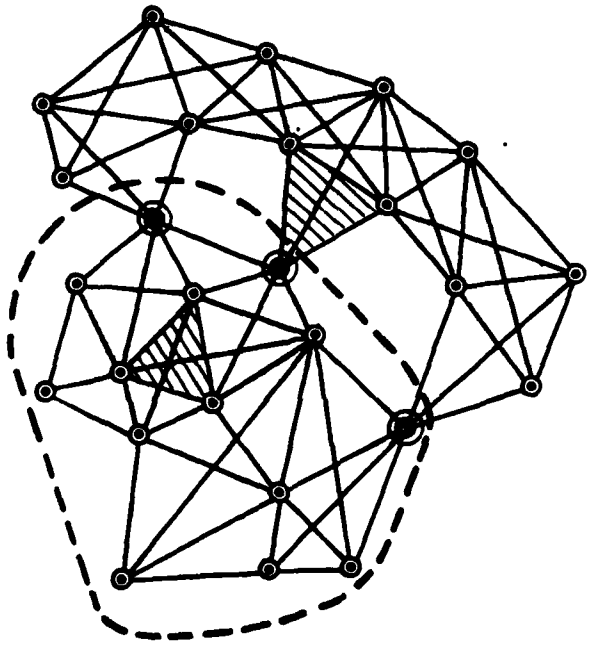


(f) 3-points are welded into the new body. Note that the only freedom of motion between the bodies is reflection with respect to the common edge (bold).

Figure 9.1 Incremental "welding" algorithms (continued).



(g) A new edge connecting the two bodies has been identified. The freedom of reflection is eliminated. The two bodies are welded together into a single body.



(h) A new body is established, initially with one point in common with the older body. When 3 points in common have been identified, the two bodies are welded together.

Figure 9.1 Incremental "welding" algorithms (continued).

3.4 Conclusions

The incremental approach to position-location problems yields attractive solutions. We have been able to develop a significant theory from which a large number of heuristics were identified and an equally large number of plausible heuristics eliminated through counterexamples.

The theoretical results were used to develop position-location algorithms that can solve a large class of positioning problems, can identify ill-conditioned and error-sensitive problems, and can be expanded to include more sophisticated heuristics employing other intelligence than distance measurements only. Finally, the position-location algorithms can be distributed among the participating network members and be executed concurrently at many sites in a decentralized fashion.

3.5 Future research

A few important directions for a future research can be identified:

1. The set of heuristics should be expanded.

2. The existing software should be expanded to include those heuristics and to solve a larger class of problems.

3. The problem of analyzing measurement errors and eliminating them through redundant measurements should be addressed.

4. A protocol to decentralize the position location algorithms should be developed.

4. SOFTWARE TESTBED

As part of the DSN project a software testbed was implemented for performing some experiments related to the distributional nature of DSN in general.

The model used for the DSN environment was one of remote processes cooperating to achieve a common joint objective, with limited communication capability. Due to the remoteness of the processes and the limited communication bandwidth between them, no synchronization is assumed. In addition, the communication subsystem is assumed to have a random performance of the same nature and characteristics as that of a "typical" computer communication network, whose performance varies as a function of the total load on the entire network.

A set of moving targets is simulated, each with arbitrary independent characteristics (time-varying speed and heading). The set of all the targets, their momentary positions, and velocities is considered as the state of the world, which is sometimes referred to as the true-state, as opposed to the estimated-state produced by the DSN.

Several sensors are simulated, each with an arbitrary position and characteristics. A sensor is modeled as a time-dependent random filter which performs a transformation of the limited world information (to which it has access) into a set of observations.

A part of each sensor definition is its characteristics, which determine its precision (or measurements noise), its range, its detection probability distribution (which depends on the range to the target), its false-alarm probabilities, etc. The sensors process their observations according to some regime and send summaries of their observations to a "central" unit which then reports to a human operator the estimated situation (the estimated state of the world).

The performance of the entire DSN system is the "degree of agreement" between the true-state and the estimated-state of the world.

Since both the processing and the communication capabilities are limited, the choice of what is communicated, how it is communicated, and when it is communicated is of the utmost importance.

It is obvious that in general no distributed processing scheme based on the availability of partial information only, at each sensor, can outperform a centralized system which has all the desired communication and processing capabilities, because the latter can simulate the former, if so desired, but not vice versa.

Unfortunately, this choice is never available in practice, since such a centralization is neither feasible nor desired.

One of the more interesting scenarios investigated in this framework employs sensors which are intermittent, in the sense that the operability of each is a random variable.

At any moment the load on the entire comunication system and the demand for processing depend on the number of active sensors at that instance and the number of targets detected by the sensors. When this load exceeds the capacity of the system (in either communication or processing), some flow control mechanism must be introduced to protect the system resources.

The most startling result of this kind of experiment was the realization that DSNs require yet another discipline of flow control, never encountered before in conventional computer communication networks.

Typically, flow control is introduced by way of holding sources back, such that the total demand for resources does not exceed the available capacity. In a file transfer operation, for example, the transmission is held back (i.e., delayed) until resources (e.g., buffers) become available. Hence, delay is introduced in order to guarantee that the total amount of data is communicated.

In a DSN situation, the situation is different. When the load is expected to exceed the capacity of the resources, the data rate is reduced, and therefore the total amount of data is decreased in order to guarantee that no delay exceeds some allowed threshhold.

In summary: when half of the capacity is lost, all targets are tracked, but at half of the frequency only, as opposed to an FTP application which would transfer only half of the files in their entirety rather than only half of all the files. Hence, conventional flow control favors an "all-of-part" approach whereas DSN flow control should favor a "part-of-all" approach.

In that scenario, the sensors send summaries of their detection to a process which prepares this information for graphic presentation to the operator of the system. The critical resources (bottlenecks) of the system are either the communication capacity of the network, from the sensors to this unit, or its own processing capacity. A scheme to dynamically monitor the achieved performance was implemented and used to optimize the overall system performance by adapting the transmission rates of the individual sensors.

Even though the techniques developed here are not necessarily the most general and were not proven as optimal under the most general conditions, they served an important role by focusing attention on some aspects of the DSN communication of which we were not aware before.

10. INTERNETWORK CONCEPTS

Research Staff:
Danny Cohen
Jon Postel

Research Assistants:
Greg Finn
Alan Katz
Paul Mockapetris

Support Staff:
Debe Hays
Mamie Chew

INTRODUCTION

The interconnection of packet-switched computer communication networks requires careful attention to the design of internetwork level, host level, and higher level protocols. The Internetwork Concepts (INC) research project explores aspects of protocols for the interconnection of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols.

OVERVIEW

ISI addresses its effort to the issues and methodology of interconnecting computer communication networks that were not designed or implemented with the goal of internetworking in mind.

Since the essence of the problem is the variety of models, disciplines, and methodologies (in addition to the simpler differences of formatting and performance parameters), a systematic network-independent approach is sought.

Pursuing this goal, ISI has made a significant impact on ARPA internetworking. ISI suggested introducing an essential level of protocol, the *internetwork level protocol* (IN), which is network-independent and more general than any individual network protocol.

ISI's impact upon ARPA internetworking is apparent in several areas:

1. The separation of the internetworking issues from the issues of achieving a reliable, *end-to-end* transmission control protocol (TCP).

2. The ability to support a variety of *types of service* (TOS) to better accommodate the requirements posed by the higher level protocols, such as those for terminal communication, file transfer, interactive (i.e., conversational) voice, and the like.

3. The ability of the IN level to support other protocols, besides the original TCP for internetwork communication.

RESEARCH AND DEVELOPMENT GOALS

The goals of the INC project are to provide appropriate and effective designs for the primary user service applications in the internetwork environment, based on a set of host and gateway level protocols that provide the full range of service characteristics appropriate to a wide range of applications (e.g., speech, graphics, text).

The task areas are as follows:

Internetwork Communication Concepts Research explores the possibilities for using interconnected networks of packet-switched computer communication systems in new and innovative ways, especially focusing on the conceptual notions and structuring of communication protocols.

Internetwork Applications is developing an internetwork computer-message facility which appears to the user much like one of the current ARPANET computer message processing programs (e.g., MSG), but provides for communication across network boundaries, removes several restrictions on the form of message objects, and improves the efficiency of message transmission. The main result of this experiment will be a set of guidelines for the design of internetwork computer-message systems.

Internetwork Protocol Design is studying the types of service needed for use in the internetwork environment. The resulting recommendation will provide a small set of user-selectable service types which can be translated by gateways and networks into appropriate internal network capabilities.

Internetwork Communication Concepts Research

This task identifies concepts in the use of packet-switched computer communications in the internetwork environment.

Approach

This is a difficult task that requires an awareness of both communication problems and the current and potential capabilities of packet-switched internetwork systems. New concepts for innovative use of technology often arise from discussions with people from different technical backgrounds or with different approaches to similar problems.

To carry out this task, we maintain current knowledge of the capabilities and prospects of the packet-switching internetwork system by reviewing new documents and attending meetings of the active workers in this field. We are also engaged in discussions with people who have communications problems that might or might not be relieved through use of packet-switching technology.

Any requirement to translate communication primitives between different networks may cause information loss because no network can be assumed to be able to capture the entirety of the information that other networks (or the union of all other networks) carry. The essence of our approach is to circumvent this information loss by encapsulating the information at a level which does not suffer such translations, the *internet level*.

This approach can also be applied to other services that must be provided across network boundaries but that were originally implemented in each network in a specific way, incompatible with other networks. Examples of such services are file transfer and computer-message systems.

ISI is tasked with demonstrating the feasibility of providing a message service across networks in a way which seems compatible to the individual networks involved yet is operational in the internetwork environment.

Results

Our interactions with the ARPA Internet Working Group have been influential in recognizing the need for a distinct internet datagram level protocol and the separation of the TCP into a recognized Internet Protocol (IN) and a strictly host-to-host protocol (TCP).

Internetwork Applications

The experiment in developing an internetwork computer-message service is under way.

Approach

The internetwork computer-message facility workings may most easily be understood by following a message item from its construction through its delivery and filing or destruction.

First, the user who wishes to send a message logs in to her home host computer and starts the user application message processing program (let's call this program user-message).

User-message will be much like existing message processing programs in the TENEX environment (for example, MSG).

In user-message, the user enters the commands to create a new message, possibly dropping into a text editor to rework certain fields of the message (the body of the message is also considered a field).

When entering the address field, the user will specify the internetwork address; this leads to a question about how to express an internetwork address, and how to allow partial address specification without ambiguity. A memo by Harrenstien suggests some ideas which we will draw on [9].

Users should be allowed to indicate destination hosts via symbolic names that may be nicknames used at the sender's site. The sending program would have to have a machine-readable table (symbolic names vs. network address) to do the proper routing.

For the Internet Working Group, we have recently made some suggestions on these issues by proposing a name syntax and a name server process [33].

When satisfied with her composition, the user enters the command to send the message. User-message now turns the message over to another program--the message sending and receiving program, which we will call post-office.

The form of the messages as stored and as communicated between programs will be structured to allow more efficient processing of fields by the message programs user-message and post-office, and to allow to be included structured information in the body of the message. The structuring concept used will be based on the NSW transmission format NSWB8, and the MSDTP [1].

The post-office program is an always-present background program (in some systems such programs are called daemons) which becomes active when presented with work to do. The post-office checks the message headers for addresses and sends copies of the message to counterpart post-offices.

The post-offices are users of communication services and do not particularly care how those services are implemented. From the post-office's point of view, it is engaged in interprocess communication; the fact that a network (or rather an internetwork system) is involved is irrelevant. But we care immensely about the underlying communication. The whole point of this experiment is to design mechanisms and interfaces to make it possible for the post-office to maintain its network-independent view.

The message travels across the interconnected networks and arrives at a receiving post-office.

The mechanisms actually used for communication between the post-offices are a key design parameter. When a message item is addressed to several destinations in a remote network, only one copy need be sent to the internetwork post-office for that network, and the internetwork post-office can take responsibility for routing copies to each recipient's host within that network.

This receiving post-office checks the message header fields for addresses at its site and delivers the message to the message files of those addressees. If the addressee list contains names incorrectly listed at the post-office's site, the post-office forwards the message to the correct site (if it knows it) and so informs the originating post-office.

To be able to know where to forward incorrectly addressed messages, the post-office will have to have a file of old and new address pairs; one would expect that this list would be primarily composed of message file names recently valid at this site but now discontinued.

Results

The experimental post-office is under development on TOPS-20. The design of the program and the post-office-to-post-office communication format is documented in a project internal memo [26].

Internetwork Protocol Design

This task is aimed at identifying the variety of services which users can reasonably expect from the internetwork environment and the services which different networks should offer for useful internetworking, then reviewing the host and gateway level internetwork protocols and recommending ways of including in them the service features identified.

Plan of Attack

Our approach is to examine the service offered by several networks and the service optimal to several applications, and by comparing and contrasting those findings to characterize the key aspects of service that can be reasonably provided to and selected by the user.

The type of service is important both for the end-to-end communication level and for the transmission level. Since different networks possibly offer different services, the type of service information should be available to each gateway participating in the actual transmission.

A prime characteristic is a *reliability/speed* trade-off. The user might indicate that he prefers that message losses not be recovered if this is associated with a high delay, or inversely, that *any* delay is justified in order to transmit all portions of the communication reliably. The former type of service is typical of highly redundant communication, like voice communication, and the latter is typical of communication which is not redundant at all, such as executable program files.

Another characteristic is *flow-rate*. Along this spectrum, several values may be specified, like *flood* (e.g., file transfer), *drops* (e.g., terminal communication), or *stream* (e.g., rate-bounded real-time interprocess communication).

In addition, another characteristic is *priority* to show the importance of delivery. This could influence the allocation of transmission resources, especially in time of crisis.

Results

We have defined a set of type-of-service parameters for use in the internet protocol [28], and we are working to produce a set of mappings from those parameters to specific settings for each of the networks in the internet system.

REFERENCES

[1] Haverty, J., "MSDTP--Message Services Data Transmission Protocol," RFC 713, NIC 34739, April 1976.

[2] Cohen, D., "Internetting or Beyond NCP," USC/Information Sciences Institute, IEN-11, March 1977.

[3] Postel, J., "Extensible Field Addressing," USC/Information Sciences Institute, RFC-730, NIC-40400 IEN-16, May 1977.

[4] Postel, J., "TCP Meeting Notes - 14 & 15 July 1977," USC/Information Sciences Institute, NIC-29449, August 1977.

[5] Postel, J., "Internet Meeting Notes - 15 August 1977," USC/Information Sciences Institute, IEN-3, August 1977.

[6] Postel, J., "Comments on Internetwork Protocols and TCP," USC/Information Sciences Institute, IEN-2, August 1977.

[7] Davidson, J., W. Hathaway, J. Postel, N. Mimno, R. Thomas, and D. Walden, "The ARPANET TELNET Protocol: Its Purpose, Principles, Implementation, and Impact on Host Operating System Design," *Proceedings of the Fifth Data Communications Symposium*, ACM/IEEE, Snowbird, Utah, September 1977.

[8] Cohen, D., "Issues in Transnet Packetized Voice Communication," *Proceedings of the Fifth Data Communications Symposium*, ACM/IEEE, Snowbird, Utah, September 1977.

[9] Harrenstien, K., "Field Addressing," ARPANET message, October 1977.

[10] Postel, J., "TCP Meeting Notes - 13 & 14 October 1977," USC/Information Sciences Institute, October 1977.

[11] Cerf, V. and J. Postel, "Specification of Internetwork Transmission Control Program -- TCP (Version 3)," USC/Information Sciences Institute, IEN-21, January 1978.

[12] Cohen, D., "On Names, Addresses and Routings," USC/Information Sciences Institute, IEN-23, January 1978.

[13] Postel, J., "TCP Meeting Notes - 30 & 31 January 1978," USC/Information Sciences Institute, January 1978.

[14] Postel, J., "Internet Meeting Notes - 1 February 1978," USC/Information Sciences Institute, IEN-22, February 1978.

[15] Postel, J., "Draft Internet Protocol Specification (Version 2)," USC/Information Sciences Institute, IEN-28, February 1978.

[16] Cohen, D., "A Protocol for Packet Switching Voice Communication," Computer Networks Protocols Symposium, Liege, Belgium, February 1978.

[17] Postel, J., "INWG FTP Note," USC/Information Sciences Institute, April 1978.

[18] Postel, J., "Internet Meeting Notes - 1 & 2 May 1978," USC/Information Sciences Institute, IEN-33, May 1978.

[19] Postel, J., "NSW Transaction Protocol (NSWTP)," USC/Information Sciences Institute, IEN-38, May 1978.

[20] Postel, J., "NSW Data Representation (NSWB8)," USC/Information Sciences Institute, IEN-39, May 1978.

[21] Postel, J., "TCP Meeting Notes - 15 & 16 June 1978," USC/Information Sciences Institute, June 1978.

[22] Postel, J., "Draft Specification of the Internetwork Transmission Control Protocol - Version 4," USC/Information Sciences Institute, IEN-40, June 1978.

[23] Postel, J., "Draft Internet Protocol Specification (Version 4)," USC/Information Sciences Institute, IEN-41, June 1978.

[24] Postel, J., "Latest Header Formats," USC/Information Sciences Institute, IEN-44, June 1978.

[25] Clark, D. and D. Cohen, "A Proposal for Addressing and Routing in the Internet," USC/Information Sciences Institute, IEN-46, June 1978.

[26] Postel, J., Alan Katz, Greg Finn and Paul Mockapetris, "The ARPANET Mail System," USC/Information Sciences Institute, INC Project Memo, June 1978.

[27] Cohen, D., "Some Thoughts about Multiplexing Issues in Networks," USC/Information Sciences Institute, IEN-52, August 1978.

[28] Postel, J., "Internet Meeting Notes - 2 & 3 & 4 August 1978," USC/Information Sciences Institute, IEN-53, August 1978.

[29] Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC-750, NIC-45500, September 1978.

[30] Postel, J., "Internet Protocol Specification (Version 4)," USC/Information Sciences Institute, IEN-54, September 1978.

[31] Postel, J., "Specification of the Internetwork Transmission Control Protocol - Version 4," USC/Information Sciences Institute, IEN-55, September 1978.

[32] Postel, J., "TCP Meeting Notes - 18 & 19 September 1978," USC/Information Sciences Institute, October 1978.

[33] Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN-61, October 1978.

11. PACKET RADIO TERMINAL SYSTEM EVALUATION

Staff:
Tom Ellis
Steve Saunders

INTRODUCTION

ARPA is currently developing a Packet Radio Network (PRN) that will provide a wideband data communication capability much like the ARPANET but with the added dimensions of mobility and dynamic configurability. As this concept gains acceptance in the military services, fundamental choices will need to be made about mobile PR terminals for use with the system.

This study, begun in June 1978, addresses the terminal characteristics desirable at the user level through the lower level protocol design decisions, the kind of interfaces required to support them, and the impact of such issues on terminal designs and other portions of the system. The work is intended to result in a demonstration level portable terminal to test and evaluate various solutions to the issues raised by extreme portability in the packet radio environment.

We feel that a comprehensive scientific study of these issues is important for the following reasons:

1. To properly exploit, match, and couple terminal design to the communications attributes and special capabilities of the PRN, such as broadcast, etc.

2. To determine those user interface characteristics most likely to support a wide volume of major user needs in a mobile tactical environment.

GENERAL REQUIREMENTS

The study is based upon a display terminal with voice, keyboard, and stylus or pointing input.

Display

Traditional (and well founded) approaches to permit a terminal user to utilize data more quickly and effectively use displays with sufficient capacity and resolution to provide enough surrounding context for better understanding or to provide a whole scene to enable the viewer to exercise his own analytic powers. This technique is effective for text, data, and graphics information.

Graphics

It is likely that communication of maps and charts will be desirable on a portable terminal via the PRNet. In two-way communication of maps or charts, it would be desirable to be able to communicate marks or even hand-rendered tracks directly on the map; thus we see a need for a trackable stylus to be used in conjunction with the display.

Voice

Voice, of course, is a very desirable communication mode to include, since it is especially easy to use and is an important medium in a crisis.

Text

A full alphanumeric display should be included, since text is the easiest medium to make the most explicit (again, especially valuable in a tactical environment).

In summary, we envision a mixed-media tactical terminal including the following:

Text input and output.
Graphics input and output.
Voice communication.

RELEVANT ISSUES

Here we will briefly discuss various issues relating to eventual terminal design and their integration into a useful communications environment.

Systems Considerations

Even though this study will concentrate on those issues affecting terminal designs, we expect it to raise and clarify systems-wide issues that will be valuable inputs to companion work in the total PRNet program.

The introduction of a highly portable terminal with text, graphics, and voice capabilities into a communication system such as PRN can be expected to have consequences at all levels. For example, experience transmitting speech digitally over a packet-switched communication system (the ARPANET) has shown the undesirability of a communications protocol that enforces acknowledgment and retransmission on failure at intervening nodes because of the transmission delays this introduces. In the ARPANET, a new packet type that does not require acknowledgment was introduced to support speech more effectively and reduce network traffic; a similar protocol may be needed for PRN. The specific resolution of this issue will have to take into consideration many parameters unique to PRN: realizable bandwidth, higher packet losses, and retransmission due to broadcast interference. Another issue raised by speech is the maximum packet size, since reducing packet size tends to reduce reconstitution delays (with a corresponding reduction in buffer memory required at repeaters and at the terminal) and reduce the effect of missing packets.

Another example of the impact of terminal capability on system design is graphics. Whereas speech is likely to have a primary impact on the terminal, its packet radio unit, repeaters, and stations, graphics is likely to have an impact on the host as well. One important issue that needs study is the form in which graphics information is passed over the network. Here there will be a tradeoff between transmission bandwidth utilized and processing (and possibly memory) capacity at the terminal. To minimize cost of additional logic at the terminal it may be necessary to send graphics data as a complete (or partially encoded) bit map of the display; this will increase processing and/or storage requirements at the host and will increase transmission bandwidth. Decisions on graphics data encoding bear directly on display interface designs, such as optimizing for an efficient scan access versus random access.

Another design issue is the processing required to compensate for missing packets and communication outages. While the expectations are for error-free (or at least error-flagged) individual packets, it can be expected--especially in a hostile environment--that entire packets will be lost. Protocols for missing packets within large data transfers must be examined (this can be particularly important for graphics data, where redundancy is limited). If incomplete data transmissions can be used, suitable methods must be adopted for identifying and using them. Perhaps application-directed error protocols are required within the PRN protocols,

which would have implications for the host as well as the terminal. Another aspect of the same problem is directly related to the mobility of the user; as he leaves the field of reception of one repeater and attempts to communicate with another, the PRN may not be continuously available or may be busy for some time setting up new address sequences.

Terminal-System Interaction

The issue is unresolved as to whether terminals must be supported by a host processor (remote or local), since absolute dependence on such a facility in a tactical environment might not be desirable. In principle, there is no reason that terminals cannot directly communicate as an alternative, or at least as a backup mode. However, several problem areas must be explored for appropriate solutions, such as the following:

1. How does one mobile terminal user obtain the address of another and how does he identify the terminal or user in his address query?

2. What processes must be included within a receiving terminal to manage simultaneous traffic from different sources, and what effects on protocols does this imply to allow proper treatment of precedence, user alerting, and temporary suspension of one connection in favor of another? Can the terminal divide its attention under some circumstances?

In general, the situation of the "called" terminal has not been explored well, whether the call originates from another terminal or a host. This capability is virtually nonexistent in data terminal practice today, where it would be possible to allow for multiple callers. The telephone company's solution of providing a busy signal to the second caller and no indication to the callee would probably be unacceptable in crisis conditions.

It will also be desirable to include in the terminal a party line connection capability or conferencing mode.

Mixed modality problems have been addressed to some extent in existing systems that permit mixed graphics and text, but these have tended to be in very highly constrained situations. In the use of multimedia messages, it will become absolutely necessary to synchronize the various forms so that gross misinterpretations do not occur. A voice or text reference to a picture must be in the presence of that picture, not the one before or after. (For example, the messages "Place your battery at the point marked X" and "Fire at the point marked X" had better be in the presence of the right maps.)

Specifically, protocols and system control structure should be designed not to burden the user with issues of connectivity and modes of operation. The terminal design can significantly affect how much of the net and subnet protocols the user must see.

Cost Considerations

Convenient, easy-to-use and effective communications are extremely important to the missions of tactical field personnel. To this end, excellent terminal portability and functionality are important, but cost may be an equally important issue in providing information exchange to all. One well known critical mass problem in communication is that if a communicating community is not well saturated with the communication capability, usefulness is reduced significantly. A military service must be capable of properly deploying terminals to optimally meet mission demands. This study thus explores the tradeoff effects of certain functional and architectural options on the relative cost of an eventual design.

An example of such a tradeoff is the type of encoding algorithms for either voice or graphics. The more sophisticated algorithms will require more processing in the terminal while reducing bandwidth requirements in the PRNet. Predicted cost-capability tradeoffs will be studied for technologies and techniques expected to be available. These technologies will, of course, have to be evaluated as part of total system requirements, including a) the expected terminal mission functional requirements, b) the communications capabilities, and c) the remote processing capabilities available to the terminal via the communications system.

For example, the choice between CVSD and LPC voice encoding is not self-evident.

- CVSD is now available on a single chip, and produces 8-16 kilobits/sec data rates.

- LPC will probably be available in a few years on an estimated seven chips of higher complexity, but will produce 1 - 2 kilobits/sec data rates.

The higher data rates required for CVSD will need approximately eight times as much buffer memory as the data rates required for LPC to handle the same PRNet delay variability. Further study may indicate that the higher data rates associated with CVSD for something as synchronous and continuous as voice may be incompatible with a PRNet loaded with a local crisis. On the other hand, user level protocols may alleviate traffic jams on a priority basis by enforcing routine traffic to more efficient modes of communication.

Choice of graphics data protocols will affect terminal memory requirements, terminal processing, picture painting speed, and communications bandwidth. Since other functions will undoubtedly be performed on map data bases for other purposes and devices, we feel that the functions performed for the terminals in question--such as map section selection, clipping, scaling, adding special overlay data, and resolution limiting--are the proper province of a processor at the map source and should be done with device-independent data encoding and data protocols. However, the final encoding and encapsulation of data to be transmitted to the terminal will have to be designed with several things in mind, such as ease of translation, communication efficiency, adequacy of expression, and--most important--terminal processing and display capabilities. Terminal design specifications should be carefully chosen not to run counter to the considerations mentioned above and thus force uncomfortable compromises.

At this time it appears that the driving system design problem is how to balance the requirements for a small, lightweight, portable terminal against the extensive services that the terminal is to provide and still offer these services at a low per-terminal system cost. Total systems costs are recognized to be very sensitive to terminal costs (and its immediate connection costs), since terminals will be by far the most numerous component in the system. Each feature (such as a graphics display, positional input, or voice) imposes additional requirements for processing capacity, memory, and special interfacing.

STATUS

Funding for this program commenced in June 1978. Since that date we have begun to identify technological areas that will need early exploration to resolve their impact on portable terminal designs.

Because one of the goals is to produce a demonstration level portable terminal for test and evaluation purposes, the project has begun to investigate technologies that would have significant impact on size and power requirements but still satisfy the early conception of the general functional requirements.

Digital Support Technology

CMOS digital technology, readily available, has low enough power requirements to satisfy the digital circuitry requirements in a portable terminal. Packaging density requirements can be met by utilizing the hybrid circuit substrate chip mounting techniques available from several specialists in this field. Higher density chips are expected to be available for future production terminals.

Display Technology

Considering present technology, an adequate display is the most critical functional capability. We feel that the display choices available at the time of terminal design will make a very large impact on the functional and applications roles for which the terminal design would be oriented. Further, capabilities in the display area will have a major effect on the way this project pictures the issues associated with terminals in a tactical environment.

Therefore, the project's initial thrust has been to define the desirable characteristics of a display and to identify and push appropriate technologies towards those characteristics. We postulate that for size purposes we would look for a flat panel technology.

It is likely that much of the traffic that a field unit must support would be between a base unit and the field unit. It is also reasonable to assume that much of that traffic would be in the form of text messages, including tables and forms for which compatible display formats between the base and field units would be desirable.

Since power and size restrictions would be relaxed somewhat for base units, they might well be similar in characteristics to the present popular 80 character line by 24 line capability that the standard T.V. format easily accommodates.

The minimum readily readable matrix to accommodate the above format would be based upon 5x7 dot characters in 6x10 blocks, producing a requirement for a total display of 480 pixels wide by 240 high.

Mockups of this layout were produced at several different sizes to subjectively select a minimum total display area (for portability) which is still readily readable by well sighted people. (We assume field personnel would have reasonable eyesight.) Subjectively, a reasonable print size is produced at about 100 pixels per inch resulting in a total display area of 4.8 inches wide by 2.4 inches high. Figures 1 and 2 illustrate approximately how such a display would look with text and an example of graphics. The graphics resolution appears to be satisfactory for simplified map sections. Further study is in order to determine how well this resolution will support other graphics examples.

Several display technologies were investigated to determine whether there are reasonable upcoming capabilities to support this application. Initially concentrating on just the power consumption issue, we can reject many of the technologies having power needs from one to three orders of magnitude higher than desirable. These were: light emitting diode arrays, gas discharge panels,

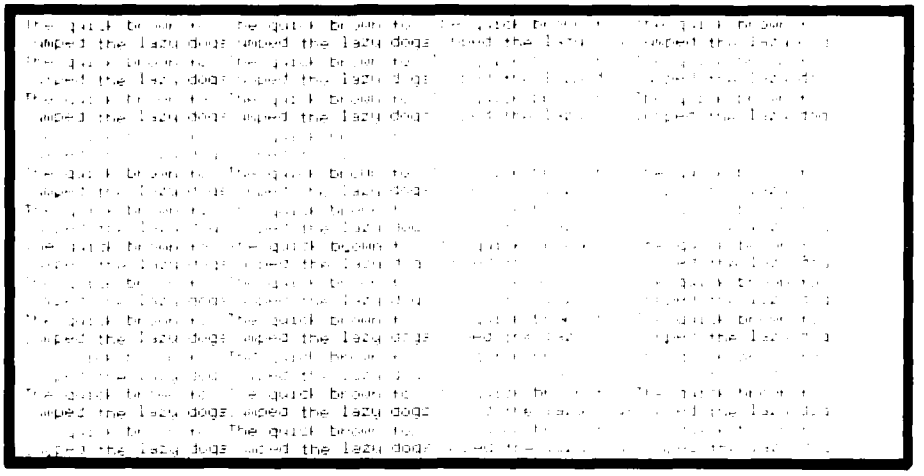


Figure 11.1 One-to-one reproduction of proposed pixel resolution: text example.

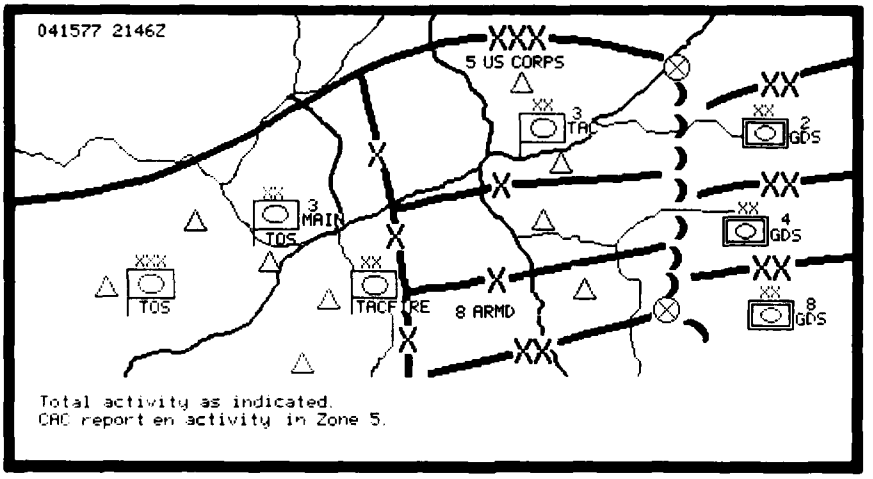


Figure 11.2 One-to-one reproduction of proposed pixel resolution: graphics example.

cathodoluminescence, electrochromics, and electroluminescence. Though the latter two may hold some promise when developed further, they are not expected to really achieve the ultra low power desired. Survivors of this contest seem to be liquid crystal and electrophorectics. Though much less mature in development, the electrophoretic (EP) materials promise to provide the best combination of ultra low power requirements and excellent viewing characteristics. The viewing angle of EP approaches that of print on paper, in contrast to the very restricted viewing angles associated with liquid crystal displays. We feel that this is an important aspect in the use of a moderately large area display (as compared to a wrist watch or calculator). Very high contrast ratios (>40:1) have been achieved with EP which further enhance its viewing characteristics.

The major unresolved issues with respect to the EP approach are as follows:

1. A fine grain array (100 points per inch) has never been tried in EP. Can techniques be developed that will prevent particle migration at this display dot size?

2. The EP medium does not possess a significant inherent threshold; therefore matrix point selection techniques must be developed.

Two subcontracts have been let to laboratories with expertise in the two respective technical areas.

Xerox PARC, Palo Alto, California, has worked independently for several years on the chemical and physical properties of electrophoretic materials. They have been contracted under this project to ferret out the problem areas and refine techniques for handling problems peculiar to this application. Their highest priority task is to investigate and refine solutions to the particle migration problem.

Panel Displays, Incorporated, has been subcontracted to fabricate arrays of thin film transistors to address and drive the EP materials.

Both of these technical efforts are too young to report progress, other than that the efforts have been launched and work is in progress.

GENERAL ISSUES

The project is now fully staffed for its first phase of work and work is in progress to explore, clarify, and consolidate the many issues which will impact the design, cost, and functionality of the portable terminals for a mobile PRNet environment.

In this study we examine a number of system design dimensions expected to have a significant impact on the tradeoffs among cost, portability, and performance. A number of dimensions have already been identified; as the study progresses we expect additional ones to emerge. Terminal system issues currently identified are as follows: the user interaction mechanism (text, graphics, voice); the partitioning of user-support processing (at the terminal, in an associated "smart box," or at the host); the connectivity of the terminal nodes (single connection of a terminal with a host or another terminal, or multiple connection of terminal or host with a set of terminals); priority interrupts to a terminal and resolution of multiple traffic sources (sometimes beneficial or complementary, sometimes disruptive or confusing). The interactions among design parameters will be evaluated in terms of their impact on such system properties as the amount of local and central processing capability needed; the requirements for interfacing the keyboard, display, graphics, and voice components to the terminal; conventions and protocols for communication and data; and required bandwidth. Other system design issues involve reliability (fail-safe/soft features, aids to remote maintenance, self-test and reporting features), security (authentication, key distribution, key loading, encryption), current and developing technologies (low-power logic, component packaging and partitioning, environmental considerations), and human factors aspects (for both the terminal equipment and the system).

12. USER-DEDICATED RESOURCE

Project Staff: Chloe Holg

BACKGROUND

The ARPA/IPTO research community's continuous expansion of the frontiers of information processing technology is reflected in a constant expansion and modification of information tools available on the ARPANET. These tools tend to have inadequate documentation (not a researcher's primary concern); moreover, because of their large numbers and many subsequent modifications, it is difficult even to keep track of them all.

ARPA has been introducing new users to the network at an increasing rate for several reasons. First, the network is a superb communications medium. Second, ARPA must transfer the results of its research programs to other agencies, and does so partly by educating selected users in the use of new information processing tools via the ARPANET.

THE PROBLEM

Before this project was established in 1977, the full benefits of network usage were not being fully realized for either ARPA or the military services. Users faced an initial barrier, partly because many were new to on-line computer systems in general, partly because system modification was so rapid, and certainly because up-to-date introductory documentation was lacking. (It is hard to consider objectively the merits of a new technology when one has difficulty getting through a TIP and logging in to an ARPANET Host.) The documentation that did exist, prepared by researchers or programmers for their own use, slighted the needs of nonprogrammer users.

THE SOLUTION

The best solution was therefore to make a single individual responsible for helping new users overcome that initial barrier by keeping track of relevant developments and communicating clearly (i.e., with minimal computer jargon). This user resource was introduced to the ARPANET community in 1977.

Although new users eventually become experienced users, there is no end to the problem: newer users are constantly introduced because of changes in the IPTO program audience and the normal turnover in military personnel. These new users are contacted as soon as their accounts are installed on the ISI machines; they are interfaced to the available network facilities by means of three levels of appropriate documentation.

Both short-term solutions (answering questions, solving individual problems) and long-term solutions (guiding users in solving common problems, making available better procedures and documentation) are routinely provided. User problems are analyzed as they occur and appropriate action decided upon. Useful inputs are also provided to system programmers to better meet the users' needs.

With ongoing efforts among the system support groups to effect standardization of resources available in TENEX and TOPS-20 across the ARPANET machines, this task has been and is yet providing broad support to users across the entire network. Military organizations provided with assistance during the lifetime of this project include the Advanced Command and Control Architecture Testbed groups; the Systems Development Laboratory and the Naval Ocean Systems Center; the Naval Research Laboratory; and Rome Air Development Center.

DIRECT USER ASSISTANCE

In this reporting period the concept of assistance to users was expanded to include direct on-site tutorial assistance to military users at the military bases themselves. Trips were made to the U. S. Army Signal School at Fort Gordon, Georgia, CACDA at Fort Leavenworth, Kansas, and TRADOC/PCAR at Fort Bragg, North Carolina, to provide on-site instruction to personnel in the use of the ARPANET in connection with the Army Data Distribution System (ADDS) program.

In June 1978 the BAA III Conference was held at Fort Gordon. A great deal of effort was spent in developing a training strategy and introductory instructional material for Signal School personnel who had been designated as participants in the ARPA/IPTO element of the conference.

DOCUMENTATION

This project has filled a long-standing need for simple, understandable, and above all *new-user-oriented* (rather than system-oriented) introductory documentation. The first major task for this project was to produce *The Joy of TENEX* [1], a TENEX manual for novice users, since available TENEX manuals

were simply unsuitable for that purpose. Both the rapid distribution of the manuals and the appearance of new documentation (e.g., *The XED Beginning Instruction Manual* [2]) necessitated a second iteration in April 1976 and a third in November 1976. Further, it became necessary to focus attention on the rather impressive set of new users accessing the ARPANET via the TIP for purposes of mail-handling, and a more basic, primer-like manual [3] was published in April 1977. The fourth version of *More Joy of TENEX* was completed, published, and distributed in 1977.

An innovative primer [4] explaining SIGMA, the operating system of the Military Message Experiment at Camp Smith, Oahu (see Section 2), was delivered to CINCPAC users in December. The *SIGMA Primer* is the first of a series; a second issue reflecting changes to the SIGMA system is planned for release in mid-1979.

As mentioned above, instructional documentation was prepared for Army Signal School Participants in the BAA III Conference at Fort Gordon in June 1978. A three-part set of TOPS-20 operating system documents [5] consisting of an Information Memorandum, a Training Manual, and a Reference Manual has also been completed and distributed for review by ADDS personnel. This project is also making a major contribution to the development of the training techniques and strategies to be employed during the critical first phase of the ADDS experiment.

FUTURE WORK

Work continues on consolidating the two ISI TENEX manuals and combining this document with TOPS-20 material to create a comprehensive two-volume set for new users [6]. An update of an on-line Network Directory program and continued development of an on-line TENEX user help system is still in progress.

As time permits, work will proceed on an update of the ISI ARPANET/TENEX/MSG Primer to include basic instruction on use of the XED text editor.

REFERENCES

1. Holg, Chloe, *The Joy of TENEX.;3...the basics* and *More Joy of TENEX.;4...some of the refinements*, Information Sciences Institute, November 1976.

2. *The XED Beginning Instruction Manual*, Information Sciences Institute, ISI/TM-76-3, May 1976.

3. Holg, Chloe, *ARPANET TENEX Primer and MSG Handling Program*, Information Sciences Institute, ISI/TM-77-4, April 1977.

4. Holg, Chloe, *ARPA/Navy/CINCPAC Military Message Experiment SIGMA Primer*, Information Sciences Institute, ISI/TM-77-9, November 1977.

5. Holg, Chloe, *ADDS Experiment Information Manual, ADDS Experiment Training Manual, ADDS Experiment Reference Manual* (forthcoming).

6. Holg, Chloe, *Joy of TENEX and TOPS-20*, Volume I, and *Joy of TENEX and TOPS-20*, Volume II (forthcoming).

13. ARPANET TENEX SERVICE

Technical Staff:

Marion McKinley, Jr.
Wanda N. Canillas
Dale Chase
Philip Crowe
Vernon Dieter
George Dietrich
Dwain Durden
Glen W. Gauthier
Kyoo C. Jo
James T. Koda
Kyle P. Lemons
James M. Lieb
Donald R. Lovelace
Raymond L. Mason
John P. Metzger
William H. Moore
Edward D. Mortenson
Robert Parker
Clarence Perkins
Vernon W. Reynolds
Dale S. Russell
Marilynne A. Sims
Barden E. Smith
Dennis Smith
William E. Stover
Lee P. Taylor
Phyllis N. Taylor
Leo Yamanaka

Consulting Staff:

David B. Fralick

Support Staff:

Larry M. Akana
Carol Carreon
Larry Fye
Oralio E. Garza
James Griffin
Richard E. Kaiser
Taylor W. Kidd
Archer J. Lewis
Robert Logsdon Jr.
Keith D. Miles
Steven R. Pollak
Randolph A. Reidel
Robert W. Robbins
Ronald L. Ross
Gary Seaton
Ronald D. Shestokes
Scot T. Smith
Michael E. Vilain
Patrick Wieber
Deborah C. Williams

INTRODUCTION

The ISI ARPANET TENEX service project presently consists of two computer centers: a local and a remote installation. The former is operated as a nonclassified developmental and service center in support of a broad set of ARPA requirements, ARPA projects, ARPA contractors, and military users. It currently services more than 2000 directories, some of which are multiplexed by several users. Approximately 95 percent of the users access the facilities via the ARPANET from locations extending from Europe to Hawaii. The latter is operated in a classified environment as part of the Advanced Command and Control Architecture Testbed (ACCAT) at the Naval Ocean Systems Center (NOSC), San Diego, California. It currently services ARPA and Navy contractors involved in the joint ARPA and U.S. Navy Command and Control experiment. The classified computer center is presently acccessible only by those personnel physically located within the classified facility. Future plans will allow remote users access via a secure ARPANET communication technique. On October 1, 1978, ISI assumed

responsibility for a second remote installation. This center is also operated in a classified environment as part of the Military Message Experiment (MME) at CINCPAC Headquarters, Camp Smith, Oahu, Hawaii.

The local computer center consists of four large-scale Digital Equipment Corporation (DEC) central processors (one KI-10, one KL-1090T and two KA-10s), Bolt Beranek and Newman (BBN) virtual memory paging boxes, large-capacity memories, on-line swapping and file storage, and associated peripherals (see Fig. 13.1). All of the above-mentioned systems presently run under control of the TENEX (originally developed by BBN) or the DEC TOPS-20 operating system, which supports a wide variety of simultaneous interactive users. In addition, the local facility supports other processors, such as several DEC PDP-11/40's, one DEC PDP-11/45, and associated peripheral devices.

The NOSC remote center consists of three large-scale DEC central processors (one KL2040T, one KA-10, and one PDP-11/70), virtual memory paging box, large capacity memories, on-line swapping and file storage and associated peripherals (see Fig. 13.2). The KA-10 runs under the TENEX operating system, the KL-2040T under the DEC TOPS-20 operating systems, and the PDP-11/70 under Western Electric Laboratories' UNIX operating system.

The MME remote center consists of one large-scale DEC KL-10B central processor, large capacity memories, on-line swapping and file storage, and associated peripherals (see Fig. 13.3). This system runs under a specially modified version of the TENEX operating system.

HARDWARE

New hardware acquired during the past year for the local computer center includes an additional 256K of core memory from Ampex. This memory has been added to ISIA, thereby increasing the memory capacity of this system to 512K. Three additional RP06 disk drives and one additional RH-20 channel interface were purchased from DEC and installed on the ISIE system to provide additional file storage and swapping speed to keep up with the ever-increasing demands by the users of this system. A distant Host interface has been installed in IMP 52. This interface, in conjunction with a BBN Host Interface and a Collins BCR which has been installed on ISIA, will allow ISI to participate in providing service to those users who will be using this system to encrypt/decrypt data according to the recently developed NBS encryption/decryption algorithm.

Also included within the ISI local computer center are two BBN H-516 Interface Message Processors (IMP), one DEC PDP-11/40 and Xerox Graphics Printer

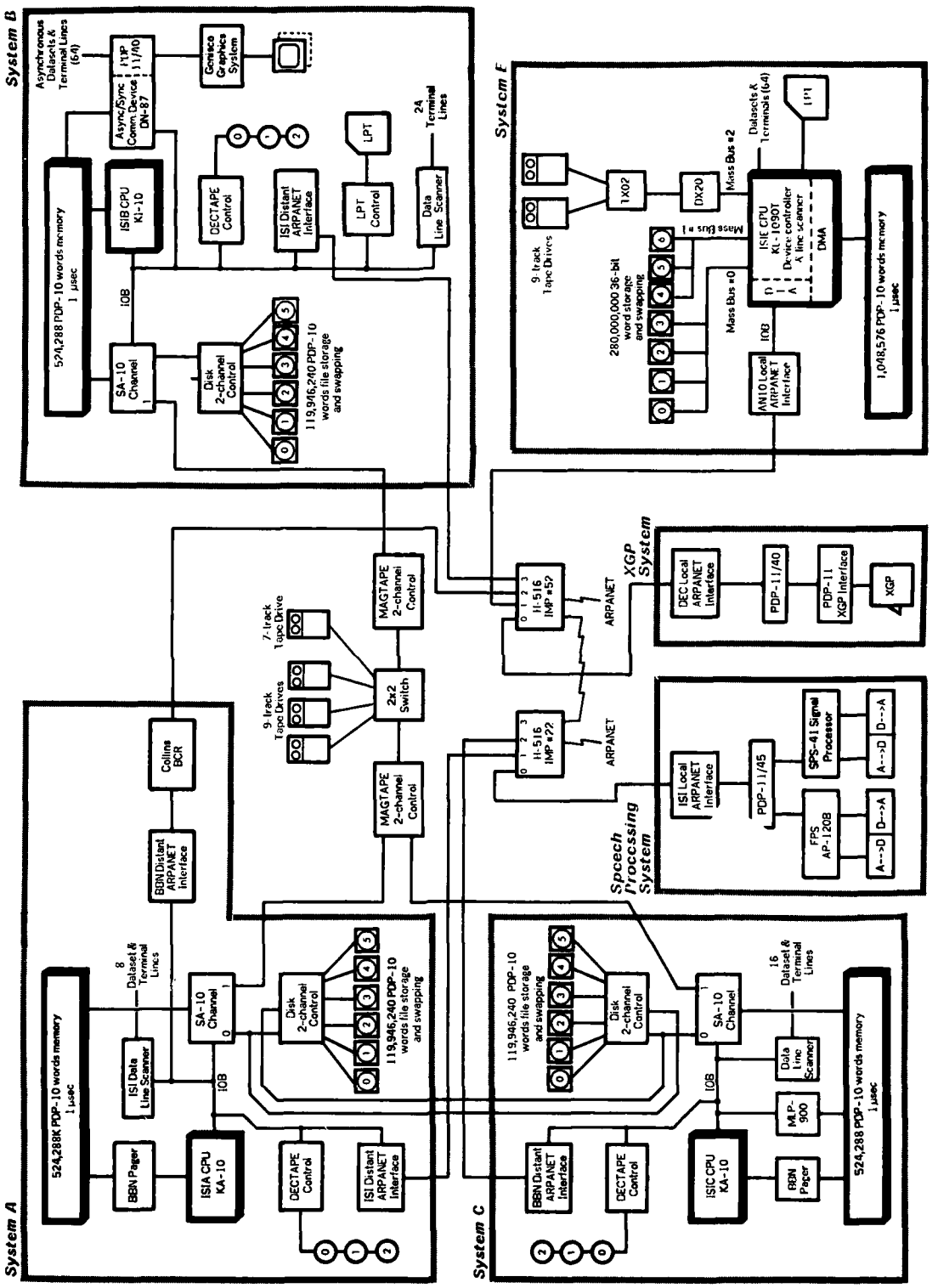


Figure 13.1 Diagram of local ISI ARPANET TENEX service facility.

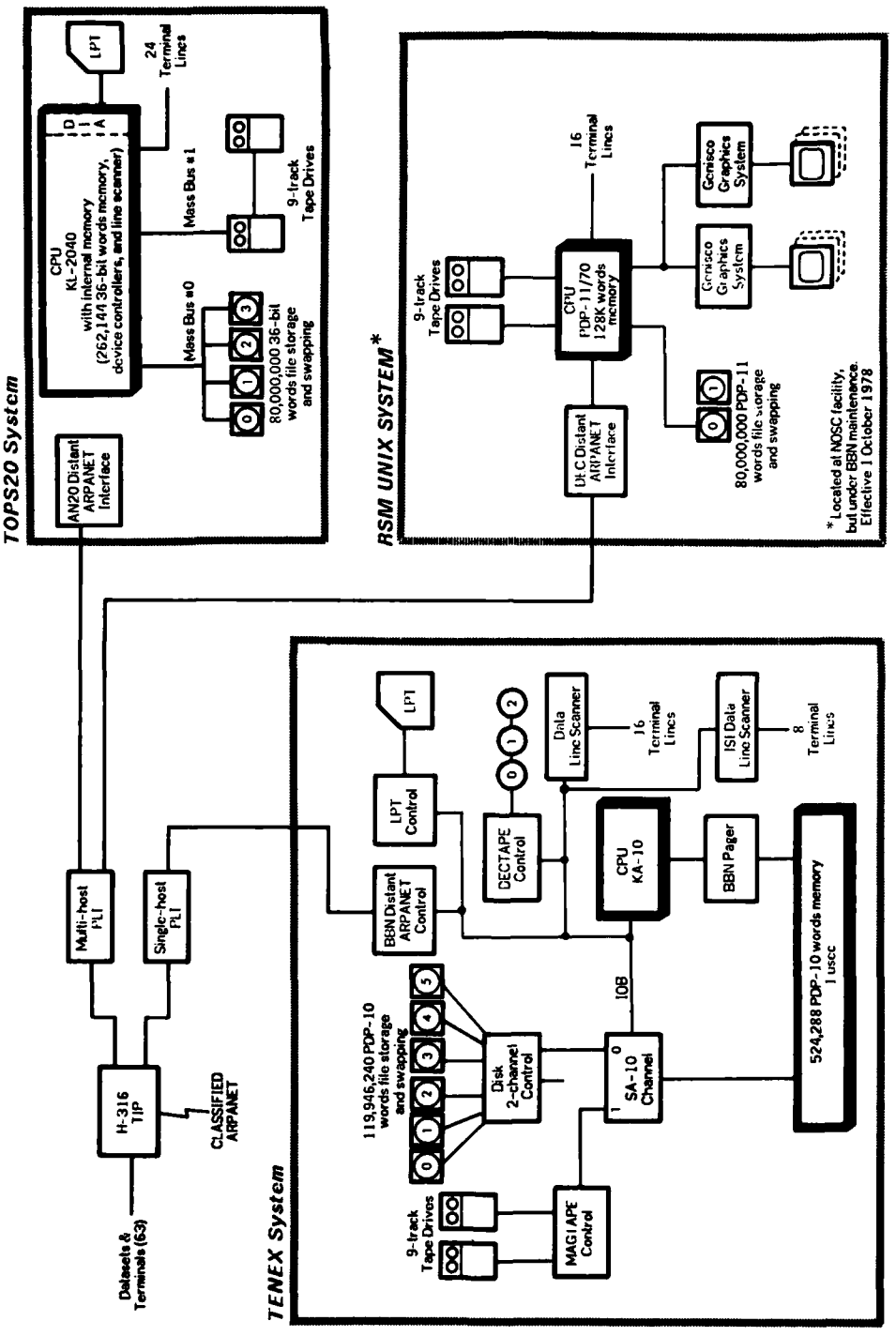


Figure 13.2 Diagram of remote ISI ARPANET TENEX service facility at NOSC.

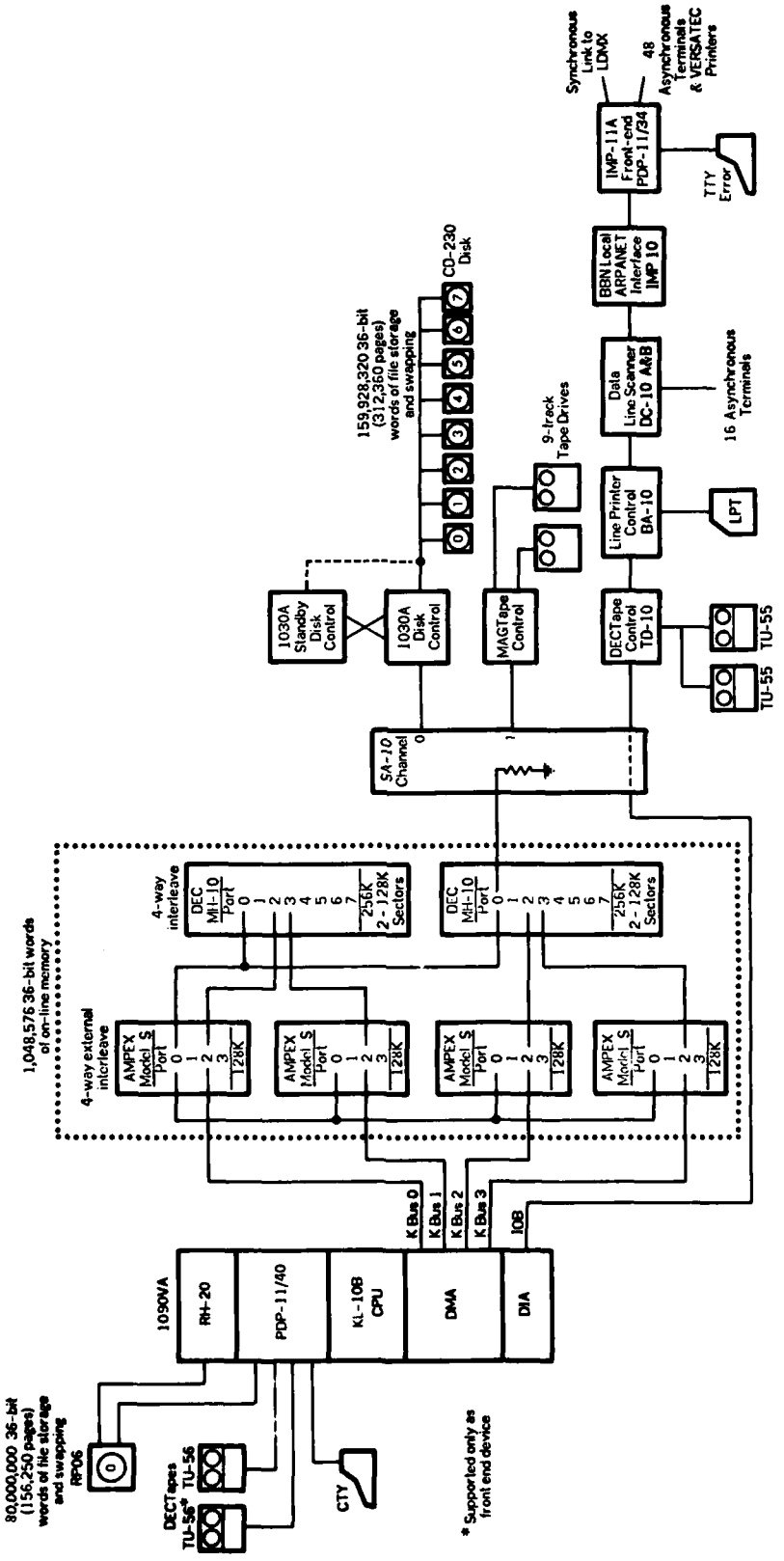


Figure 13.3 Diagram of remote ISI ARPANET TENEX service facility at Camp Smith, Oahu.

(XGP), one DEC PDP-11/45 with an SPS Signal Processing System, and a Floating Point Systems AP-120B (FPS) (configured as a speech processor), one Microprogrammable Processor (MLP-900) and several associated peripheral devices such as disk, memories, terminals, etc. from various manufacturers and several special interfaces designed and developed by ISI.

Systems ISIA and ISIC are currently designated as priority systems and are therefore cross-connected (cabled) in such a manner so that if one system crashes or is otherwise unavailable because of hardware/software maintenance or development, the system may be started as a back-up replacement system and service continued after a brief (15 minutes or less) delay to switch the file storage media and one cable.

New hardware acquired during the past year for the NOSC remote computer center includes 256K of Ampex core memory, which was added to the NOSC-KA-TENEX system, thereby increasing the memory capacity of this system to 512K. A Calcomp CD230 dual disk drive unit has been added to this system to provide increased on-line file storage and swapping. A data line scanner designed and developed by ISI has also been added to expand the number of directly connected users of this system to twenty-four. One additional RP04 disk drive purchased from DEC has been installed on the NOSC-PDP-11/70 Unix system to provide either backup or additional file storage for this system. One eight-line communications group was installed on the NOSC-KL-2040-TOPS-20 system. Attached to this line group is a specially designed TDM multiplexor utilizing fiber-optic lines for communication to other areas within the NOSC complex, so that remotely located users (within the NOSC complex) may use the ACCAT computer facility.

Recently, ISI was tasked by ARPA/IPTO to develop a cost-effective system that would increase throughput and response of the SIGMA (MME) message system. The acquisition of necessary equipment has already begun. ISI has been successful in negotiating a trade with Digital Equipment Corporation, exchanging many of the items (i.e., CPU, memory, etc.) in the existing MME system at CINCPAC Headquarters for credit toward the purchase of a newer state-of-the-art processor. This equipment will be augmented by additional equipment purchased from other vendors as well as many items from the existing MME system. This will result in one large-scale integrated system that will meet the aforementioned requirements. A minimally configured system is presently installed within the ISI local computer center for development of the special TENEX operating system. New hardware, either traded for or purchased, includes the following: one DEC KL-10B CPU, one DEC RH-20 channel interface with one DEC RP06 disk drive, two DEC MH-10 memories (256K each) and one Calcomp 1030A disk controller with one Calcomp CD-230 (two-spindles) disk drive. Refer to Figure 13.3 for final configuration of

this system. This system is presently scheduled for installation on or about October 6, 1978, and is expected to be operational soon thereafter.

Purchase orders have been issued to DEC for a new 2060T, TOPS-20 AN system. This system is scheduled for delivery in November 1978. It will be designated as ISID and will provide computer service to NLS users at the Air Force Data Systems Design Center (AFDSDC), Gunter Air Force Station, Montgomery, Alabama. This system will also provide computerized message and editing capabilities to users from the ARPA-sponsored Army Data Distributions System (ADDS) project at Fort Bragg, North Carolina.

SOFTWARE

During the year, the prime concern of the software group has been providing stable software at a consistent level across all ISI TENEX and TOPS-20 systems. This includes the monitor, subsystem and utility programs. Work was also done with various diagnostic programs to add new features and improve existing functions.

The TENEX monitor used on all ISI TENEX systems is version 134 from BBN, with additions of a few utility JSYS's for local support, and minor bug fixes. Many of the bug fixes were obscure anomalies in file and process support JSYS's, which came to light during the development of the SIGMA MME system. Improvements in monitor overhead were made in the area of terminal data transfer. Data transfer rates were greatly improved by taking advantage of the direct memory access feature in the DN87 terminal hardware.

The TOPS-20 monitor used on all ISI TOPS-20 systems is version 101B from DEC, with local support additions and a few minor bug fixes. An important increase in disk storage was allowed by a slight change to the internal address formats. System security was greatly improved by incorporating the password encryption present in TENEX into the TOPS-20 operating system. This algorithm transforms an alphanumeric string of up to 39 characters into a 72-bit binary pattern.

Early this year, ISI assumed responsibility for the support and maintenance of the NLS subsystem, version 8.5. Our level of support includes assistance to ARPA and AFDSDC personnel in using the features of NLS 8.5, minor bug fixes and extensions of existing functions, but excludes any significant design changes.

Subsystem maintenance is a continuous effort. The File Update Support System (FUSS), which was developed last year, has proved to be a very effective

tool for providing consistent levels of software across several machines. Additional work brought about several important enhancements. There is now an interactive interface for collecting update information, which helps eliminate typing mistakes and forgotten update criteria. Security features were developed to help validate update messages and prevent bogus entries from causing problems.

Work on diagnostic programs was undertaken to add new features and improve existing functions. Many of the efforts were directed towards the diagnostic for the Systems Concepts SA10 (DEC to IBM translator), and the peripheral devices cabled to it. Extensive portions were rewritten to improve interactions with the hardware service engineer. This included better input capabilities, informative error messages, and extended testing facilities. The rewrite also brought improved organization and modularity to the coding, improving comprehension of the various tests.

Work has commenced on two major projects. TENEX has been modified to run on a DEC 1090-VA (KL-10), which has supported the MME Project at CINCPAC Headquarters since early in the fourth quarter of 1978. The KL-10 is run in "KI-10 mode," so that KL-TENEX closely resembles KI-TENEX with necessary changes required to support hardware improvements and additions, including the paging hardware, clocks and timing, and a cache memory. Peripheral diagnostics were also changed to run on the KL-10.

The DEC TOPS-20 operating system is being upgraded to version 3A, and is expected to be released from DEC in mid-1979. Prior to installation on our TOPS-20 systems, there are several local changes which must be incorporated into the basic release. Many are bug fixes which will hopefully already be present. Others are important local changes, especially in scheduling and page management. A sizeable effort will be required in the area of subsystem modification. In this release, DEC modified specifications to several existing JSYS's (monitor calls), and this will require extensive checking and possible modification of several subsystems and user programs to insure the new specifications are observed.

SUPPORT PERSONNEL

ISI provides seven-day-a-week, twenty-four-hour-a-day operator, software, and hardware support for the local computer center. At least one operator is physically on-site at all times, and the systems programmers and computer service engineers either are physically on-site or are scheduled for one-hour on-call service. The ISI remote NOSC computer center is currently manned as a five-day-a-week, eight-hour-a-day type of operation and all support personnel are physically on-site only during these times. The ISI remote MME computer center

will be manned on a twenty-four-hour-a-day, seven-day-a-week basis. At least one operator will be physically on-site at all times, and the systems programmers and computer service engineers will be physically on-site or on a one-hour on-call schedule.

RELIABILITY

To provide required hardware/software preventive and/or corrective maintenance of the equipment, ISI will continue scheduling each of the TENEX/TOPS-20 systems as "out of service" (unavailable to users) for seven contiguous hours each week. The remaining 161 hours of each week are intended to be devoted entirely (100 percent) to user service. The actual long-term up-time for the network service machines has exceeded 98 percent of scheduled up-time for the last year.

LOCAL PROJECT SUPPORT

The local service center has been used extensively in support of local projects. The ISI staff makes use of the available standard subsystems and some staff members have written subsystems and utilities to support their own projects. The facility also supports less frequently used subsystems at the special request of users (e.g., PDP-11 cross-assemblers and the DECUS Scientific Subroutine Package).

ISI PUBLICATIONS

Research Reports

Abbott, Russell J., *A Command Language Processor for Flexible Interface Design*, ISI/RR-74-24, February 1975.

Anderson, Robert H., *Programmable Automation: The Future of Computers in Manufacturing*, ISI/RR-73-2, March 1973; also appeared in *Datamation*, Vol. 18, No. 12, December 1972, pp. 46-52.

---, and Nake M. Kamrany, *Advanced Computer-Based Manufacturing Systems for Defense Needs*, ISI/RR-73-10, September 1973.

Balzer, Robert M., *Automatic Programming*, ISI/RR-73-1 (draft only).

---, *Human Use of World Knowledge*, ISI/RR-73-7, March 1974.

---, *Imprecise Program Specification*, ISI/RR-75-36, May 1976; also appeared in *Calcolo*, Vol. XII, Supplement 1, 1975.

---, *Language-Independent Programmer's Interface*, ISI/RR-73-15, March 1974; also appeared in *AFIPS Conference Proceedings*, Vol. 43, AFIPS Press, Montvale, N. J., 1974.

---, Norton R. Greenfeld, Martin J. Kay, William C. Mann, Walter R. Ryder, David Wilczynski, and Albert L. Zobrist, *Domain-Independent Automatic Programming*, ISI/RR-73-14, March 1974; also appeared in *Proceedings of the International Federation of Information Processing Congress*, 1974.

---, Neil M. Goldman, and David Wile, *Informality in Program Specifications*, ISI/RR-77-59, April 1977.

---, Neil Goldman, and David Wile, *Meta-Evaluation as a Tool for Program Understanding*, ISI/RR-78-69, January 1978.

---, Neil Goldman, and David Wile, *On the Use of Programming Knowledge*, ISI/RR-77-63, October 1977.

Bisbey, Richard L., Jim Carlstedt, Dale M. Chase, and Dennis Hollingworth, *Data Dependency Analysis*, ISI/RR-76-45, February 1976.

---, and Gerald J. Popek, *Encapsulation: An Approach to Operating System Security*, ISI/RR-73-17, December 1973.

Britt, Benjamin, Alvin Cooperband, Louis Gallenson, and Joel Goldberg, *PRIM System: Overview*, ISI/RR-77-58, March 1977.

Carlisle, James H., *A Tutorial for Use of the TENEX Electronic Notebook-Conference (TEN-C) System on the ARPANET*, ISI/RR-75-38, September 1975.

Carlstedt, Jim, Richard L. Bisbey II, and Gerald J. Popek, *Pattern-Directed Protection Evaluation*, ISI/RR-75-31, June 1975.

Cohen, Dan, *Specification for the Network Voice Protocol*, ISI/RR-75-39, March 1976.

Crocker, Stephen D., *State Deltas: A Formalism for Representing Segments of Computation*, ISI/RR-77-61, September 1977.

Ellis, Thomas O., Louis Gallenson, John F. Heafner, and John T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, ISI/RR-73-12, June 1973.

Gallenson, Louis, *An Approach to Providing a User Interface for Military Computer-Aided Instruction in 1980*, ISI/RR-75-43, December 1975.

Gerhart, Susan L., *Program Verification in the 1980s: Problems, Perspectives, and Opportunities*, ISI/RR-78-71, August 1978.

Goldman, Neil, Robert M. Balzer, and David Wile, *The Inference of Domain Structure from Informal Process Descriptions*, ISI/RR-77-64, October 1977.

Good, Donald I., Ralph L. London, and W. W. Bledsoe, *An Interactive Program Verification System*, ISI/RR-74-22, November 1974; also appeared in *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1975, pp. 59-67.

Guttag, John V., Ellis Horowitz, and David R. Musser, *The Design of Data Type Specifications*, ISI/RR-76-49, November 1976.

Guttag, John V., James H. Horning, Ralph L. London, *A Proof Rule for Euclid Procedures*, ISI/RR-77-60, May 1977; also in Neuhold, E. J., (ed.) *Formal Description of Programming Concepts*, North-Holland Publishing Co., 1978, pp. 211-220.

Heafner, John F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, ISI/RR-74-21, September 1974.

---, *Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings*, ISI/RR-75-34, July 1975.

Igarashi, Shigeru, Ralph L. London, and David C. Luckham, *Automatic Program Verification I: A Logical Basis and Its Implementation*, ISI/RR-73-11, May 1973; also appeared in *Artificial Intelligence Memo 200*, Stanford University, May 1973 and *Acta Informatica*, Vol. 4, No. 2, 1975, pp. 145-182.

Kamrany, Nake M., *A Preliminary Analysis of the Economic Impact of Programmable Automation Upon Discrete Manufacturing Products*, ISI/RR-73-4, October 1973.

Kimbleton, Stephen R., *A Heuristic Approach to Computer Systems Performance Improvement. I: A Fast Performance Prediction Tool*, ISI/RR-74-20, March 1975.

Levin, James A., and James A. Moore, *Dialogue Games: Meta-Communication Structures for Natural Language Interaction*, ISI/RR-77-53, January 1977.

---, and Neil M. Goldman, *Process Models of Reference in Context*, ISI/RR-78-72, October 1978.

---, James A., and Armar A. Archbold, *Working Papers in Dialogue Modeling*, Volume I, ISI/RR-77-55, January 1977.

London, Ralph L., Mary Shaw, and William A. Wulf, *Abstraction and Verification in ALPHARD: A Symbol Table Example*, ISI/RR-76-51, December 1976.

Lynn, Donald S., *Interactive Compiler Proving Using Hoare Proof Rules*, ISI/RR-78-70, January 1978.

Mann, William C., *Dialogue-Based Research in Man-Machine Communication*, ISI/RR-75-41, November 1975.

---, *Man-Machine Communication Research Final Report*, ISI/RR-77-57, February 1977.

---, *Why Things Are So Bad for the Computer Naive User*, ISI/RR-75-32, March 1975.

---, James A. Moore, James A. Levin, and James H. Carlisle, *Observation Methods for Human Dialogue*, ISI/RR-75-33, July 1975.

---, James H. Carlisle, James A. Moore, and James A. Levin, *An Assessment of Reliability of Dialogue Annotation Instructions*, ISI/RR-77-54, January 1977.

---, Greg Scragg, and Armar A. Archbold, *Working Papers in Dialogue Modeling*, Volume II, ISI/RR-77-56, January 1977.

Martin, Thomas H., Monty C. Stanford, F. Roy Carlson, and William C. Mann, *A Policy Assessment of Priorities and Functional Needs for the Military Computer-Aided Instruction Terminal*, ISI/RR-75-44, December 1975.

Miller, Lawrence H., *An Investigation of the Effects of Output Variability and Output Bandwidth on User Performance in an Interactive Computer System*, ISI/RR-76-50, December 1976.

Moore, James A., James A. Levin, and William C. Mann, *A Goal-Oriented Model of Natural Language Interaction*, ISI/RR-77-52, January 1977.

Moriconi, Mark S., *A System for Incrementally Designing and Verifying Programs*, Volume I, ISI/RR-77-65, January 1978.

---, *A System for Incrementally Designing and Verifying Programs*, Appendix, Volume II, ISI/RR-77-66, January 1978.

Musser, David R., *A Proof Rule for Functions*, ISI/RR-77-62, October 1977.

Oestreicher, Donald R., *A Microprogramming Language for the MLP-900*, ISI/RR-73-8, June 1973; also appeared in the Proceedings of the ACM Sigplan Sigmicro Interface Meeting, New York, May 30-June 1, 1973.

Richardson, Leroy, *PRIM Overview*, ISI/RR-74-19, February 1974.

Rothenberg, Jeff, *An Editor to Support Military Message Processing Personnel*, ISI/RR-74-27, June 1975.

---, *An Intelligent Tutor: On-Line Documentation and Help for A Military Message Service*, ISI/RR-74-26, May 1975.

Shaw, Mary, William A. Wulf, and Ralph L. London, *Abstraction and Verification in ALPHARD: Iteration and Generators*, ISI/RR-76-47, August 1976.

Tugender, Ronald, and Donald R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, ISI/RR-74-23, May 1975.

Wilczynski, David, *A Process Elaboration Formalism for Writing and Analyzing Programs*, ISI/RR-75-35, October 1975.

Wulf, William A., Ralph L. London, and Mary Shaw, *Abstraction and Verification in ALPHARD: Introduction to Language and Methodology*, ISI/RR-76-46, July 1976; also appeared in *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, December 1976, pp. 253-265.

Yonke, Martin D., *A Knowledgeable, Language-Independent System for Program Construction and Modification*, ISI/RR-75-42, December 1975.

Special Reports

Annual Technical Report, May 1972 - May 1973, ISI/SR-73-1, September 1973.

A Research Program in the Field of Computer Technology, Annual Technical Report, May 1973 - May 1974, ISI/SR-74-2, July 1974.

A Research Program in Computer Technology, Annual Technical Report, May 1974 - June 1975, ISI/SR-75-3, September 1975.

Bisbey, Richard L., Gerald Popek, and Jim Carlstedt, *Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time*, ISI/SR-75-4, January 1976.

Carlstedt, Jim, *Protection Errors in Operating Systems: Validation of Critical Variables*, ISI/SR-75-5, May 1976.

A Research Program in Computer Technology, Annual Technical Report, July 1975 - June 1976, ISI/SR-76-6, July 1976.

Hollingworth, Dennis, and Richard L. Bisbey II, *Protection Errors in Operating Systems: Allocation/Deallocation Residuals*, ISI/SR-76-7, June 1976.

1977 Annual Technical Report: A Research Program in Computer Technology, July 1976-June 1977, ISI/SR-77-8, November 1977.

Carlstedt, Jim, *Protection Errors in Operating Systems: Serialization*, ISI/SR-77-9, April 1978.

Carlstedt, Jim, *Protection Errors in Operating Systems: A Selected Annotated Bibliography and Index to Terminology*, ISI/SR-78-10, January 1978.

Hayden, Charles, Peter W. Alfvin, and Stephen D. Crocker, *Multi-Microprocessor Emulation: Annual Report for 1977*, ISI/SR-78-12, April 1978.

Bisbey, Richard, and Dennis Hollingworth, *Protection Analysis: Final Report*, ISI/SR-78-13, July 1978.

Technical Manuals

Gallenson, Louis, Joel Goldberg, Ray Mason, Donald Oestreicher, and Leroy Richardson, *PRIM User's Manual*, ISI/TM-75-1, May 1975.

XED User's Manual: Beginning Instruction, ISI/TM-76-3, May 1976.

Holg, Chloe, *ARPANET/TENEX Primer and MSG Handling Program*, ISI/TM-77-4, April 1977.

Gallenson, Louis, Alvin Cooperband, and Joel Goldberg, *PRIM System: AN/UYK-20 User Guide/User Reference Manual*, ISI/TM-77-5, October 1977.

---, *PRIM System: U1050 User Guide/User Reference Manual*, ISI/TM-77-6, October 1977.

---, *PRIM System: Tool Builder's Manual/User Reference Manual*, ISI/TM-78-7, January 1978.

Holg, Chloe, *ARPA Navy CINCPAC Military Message Experiment SIGMA Primer*, ISI/TM-77-9, December 1977.

Oestreicher, Donald R., Paul Raveling, and Robert H. Stotz, *HP/MME Terminal Application Specification*, ISI/TM-78-10, March 1978.

Rothenberg, Jeff, *DARPA Navy CINCPAC Military Message Experiment: SIGMA Message Service Reference Manual*, ISI/TM-78-11, March 1978.